ANTEPRIMA

Fondamentali del WWW

Quello che devi sapere per fare "Web Hacking"

Stefano Novelli

Parte del progetto

ATTENZIONE!

Questa è una versione in "**Anteprima**", ovvero non revisionata e che potrebbe contenere errori grammaticali o tecnici. Se ne consiglia la lettura solo agli addetti ai lavori che già hanno competenze nel campo.

La versione definitiva verrà rilasciata sul sito www.hacklog.net

AVVERTENZE

La violazione di un computer o rete altrui è un reato perseguibile penalmente dalla legge italiana (art. 615 ter del Codice Penale). Alcune delle procedure descritte sono da ritenersi a titolo scolastico/illustrativo/informativo e messe in pratica solo su dispositivi in nostro possesso o in ambienti di test controllati, pertanto il lettore solleva gli autori di questo documento da ogni responsabilità circa le nozioni assimilate durante il corso e le conseguenze verificabili.

Quanto narrato in alcune parti di questo libro è opera di fantasia. Ogni riferimento a cose, persone o fatti realmente accaduti è puramente casuale.

NOTE SULL'OPERA

I contenuti di "Fondamentali del WWW" sono rilasciati gratuitamente per tutta la rete e disponibile in vari formati, secondo l'autoregolamentazione dell'ethical hacking e il rispetto delle realtà culturali che lo praticano.

Sei libero di poter prendere parti del documento per qualunque opera, citandone opportunamente la fonte (Hacklog di inforge.net) e possibilmente ove possibile con link ipertestuale in calce. Essendo un progetto che ha richiesto molto tempo ritengo che se il documento sia stato utile ai fini di progetti terzi venga condiviso per rispetto verso il sottoscritto, i suoi collaboratori e coloro che hanno creduto in esso.

DIRITTI D'AUTORE

GLOSSARIO

PREFAZIONE

Abbiamo deciso di rilasciare anticipatamente il documento "Fondamentali del WWW" per dare modo, a chi sta aspettando Hacklog 2, di prepararsi agli argomenti che verranno affrontati nel secondo capitolo. Verranno tenute le numerazioni come già presenti in Hacklog 2, quindi partirà direttamente dal 2.

Il documento è diviso in due parti:

2. Gli Strumenti del Mestiere

Questa parte del documento fa parte dell'Hacklog 2 e viene integrata per dar modo agli utenti di prepararsi l'ambiente di sviluppo (LAB) in cui potranno effettuare i loro attacchi.

3. I Fondamentali del WWW

Questa parte del documento spiega le nozioni base del WWW e della programmazione affinché siano in grado di comprendere i meccanismi della Sicurezza Informatica in ambiente Web.

Riteniamo che il conoscere almeno le basi di ciò che riguarda la sicurezza di uno specifico ambiente può essere l'occasione per imparare davvero qualcosa, piuttosto che seguire alla cieca dei wiki cotti e mangiati. Hacklog vuole essere un ambiente di crescita personale e professionale, non un libro di cucina ;)

Se ci dovessero essere delle incongruenze con la lettura, errori grammaticali e altro potete scrivermi a info@hacklog.net. La mail non è di supporto a problemi che possono essere tranquillamente risolti nel forum o nella pagina ufficiale Facebook dell'Hacklog.

Buona lettura,

Stefano Novelli

2. GLI STRUMENTI DEL MESTIERE

L'ambiente di sviluppo è fondamentale per comprendere sul lato pratico in che modo vengono veicolati gli attacchi informatici attraverso il vettore del WWW: consigliamo caldamente il lettore di applicare ciò che verrà spiegato per semplificare il processo di apprendimento e nello stesso tempo vivere pienamente l'esperienza che Hacklog 2 ha da offrire.

2.1 Ambiente d'Attacco

Buona parte di questo documento farà uso del Sistema Operativo **Debian 9**,[1] attualmente l'ultima versione di questa distribuzione, per il computer che useremo nelle fasi d'attacco.

Qualora decidessi di mettere in pratica le tecniche di attacco, dovrai essere già in grado di installare il seguente Sistema Operativo all'interno del tuo computer (o se preferisci tramite l'uso di una Virtual Machine). Nell'eventualità che tu non sia in grado di installare Debian 9, puoi fare riferimento al "*Manuale d'Installazione di Debian GNU/Linux*", distribuito gratuitamente sul nostro sito www.hacklog.it.

Il manuale è adatto anche per coloro che decidono di preferire una distribuzione di tipo **Pentesting**. Le distribuzioni consigliate sono tutte quelle basate su Debian e Ubuntu, ovvero:

- **Parrot Security OS** (https://www.parrotsec.org)
- **Backbox** (https://www.backbox.org)
- **Kali Linux** (https://www.kali.org)

e molte altre.

---

[1] NDA: non è sicuro che utilizzeremo Debian 9, valuteremo a fine revisione quale distribuzione sarà adeguata al tipo di esperienza che vogliamo offrire. Al momento, la guida si basa su Debian 9.

Considerata la natura delle distribuzioni madre, non offriremo supporto a comandi e configurazioni di altro tipo (Fedora, Suse, Arch, Gentoo etc…), pertanto consigliamo il lettore inesperto di ambienti GNU/Linux di astenersi dal “fai-da-te” e di utilizzare quanto consigliato.

Perché utilizzare GNU/Linux e non invece Windows/macOS?

È luogo comune che in IT Security colui che effettua pentest debba usare GNU/Linux. Non è mia intenzione “imporre” l’uso di una macchina GNU/Linux e, quando sarà possibile, cercherò anche di considerare i Sistemi Operativi di Microsoft e Apple. I motivi legati alla scelta di GNU/Linux sono da ricercare in altri fattori: comodità e popolarità.

Dire che GNU/Linux è più comoda rispetto a Windows o Mac può sembrare un eufemismo ma ti assicuro che non è poi così lontano dalla realtà: paradossalmente, installare un programma da Terminale risulta essere più semplice e standardizzato piuttosto che spiegare l’installazione di un programma negli altri due OS (per quanto facile sia da fare). Nel caso in cui si verifichi un errore, sarà molto più immediato ricercare una risoluzione ad esso, grazie all’infinita community online che ogni giorno aiuta gli users della rete.

La comodità si allaccia inoltre perfettamente alla popolarità che tale OS ha sia nel ramo dell’IT Security che in quello dei Server: nel primo caso troverai moltissimi software pensati esclusivamente per GNU/Linux – e altri tools che ne “emulano” il funzionamento, spesso malamente, su altre piattaforme – mentre nel secondo caso, e a parte in rare eccezioni o necessità, ci ritroveremo ad effettuare test d’attacco su macchine GNU/Linux. Basti vedere a quanti Hosting/VPS/Server e via dicendo che fanno uso solo del Sistema Operativo del pinguino.

Questo tuttavia non prevarica certo la conoscenza di un Sistema Operativo di diversa natura: per diversi motivi l’utente dovrebbe avere la necessità di usare uno specifico OS o di trovarsi di fronte a un Windows Server/Mac OS Server in fase d’attacco: in questi casi è sempre buona norma avere delle skill su tali Sistemi Operativi per evitare di bloccarsi sul più bello!

2.1.1 Crea la tua Macchina Virtuale

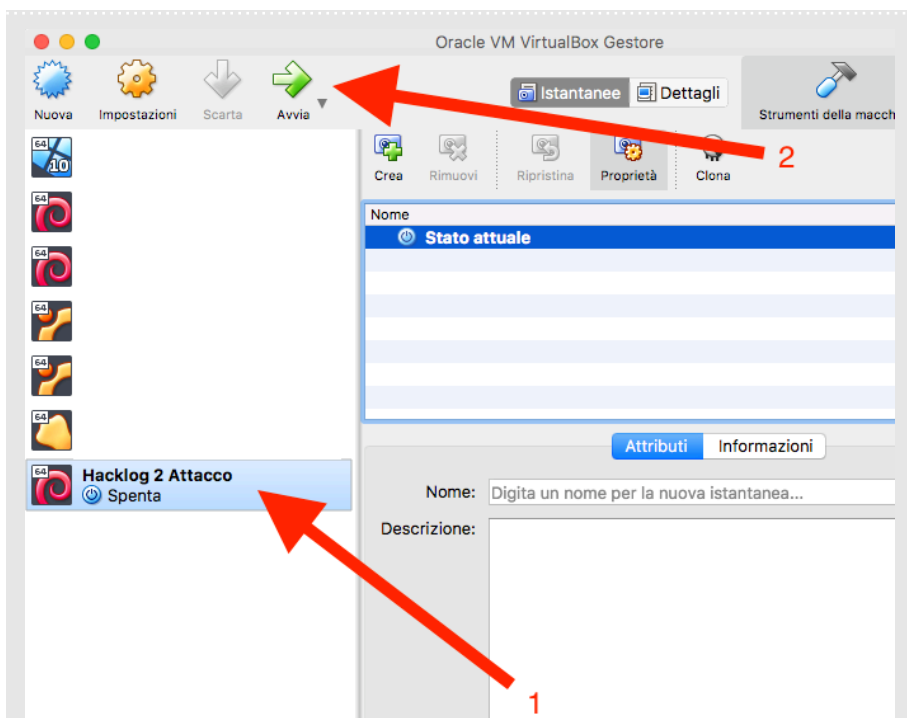
Per una maggiore comodità vorresti poter utilizzare il Sistema Operativo all'interno di una **Macchina Virtuale**: a differenza del primo volume, dove era necessario esporre la macchina d'uso in un ambiente reale, in questo volume puoi tranquillamente utilizzarne una.

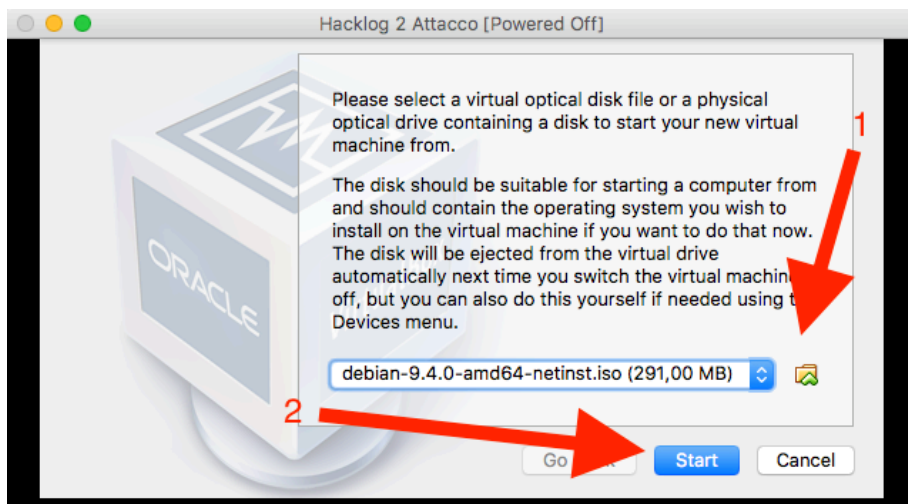
Meglio un'installazione fisica o una Virtual Machine?
In questo volume ti consigliamo di utilizzare una Virtual Machine: questa è un contenitore in grado di simulare un vero computer all'interno di un Sistema Operativo già in uso, il che si traduce nel non dover formattare e installare alcun OS rischiando così di dover cancellare partizioni sul tuo disco fisso.
Considera inoltre che effettueremo, nel corso della guida, un setup di due macchine virtuali: la prima d'attacco e la seconda di difesa. Queste verranno affiancate per darci modo di studiare gli effetti degli attacchi con un miglior controllo degli ambienti.

Per prima cosa scarica e installa Oracle VM VirtualBox[1] per il tuo Sistema Operativo, quindi clicca su *Nuovo* e crea un nuovo contenitore in cui risiederà la tua VM. Dagli un nome e una tipologia/versione (la combinazione Linux Debian 64bit andrà benissimo) [figuraX].

[1] https://www.virtualbox.org/

FiguraX: Assegna un nome e un tipo di Sistema Operativo

Definisci RAM e Spazio su Disco che vuoi dedicare alla macchina, quindi segui tutta la procedura finché non comparirà il tasto d'avvio [figuraX].

FiguraX: crea e avvia per la prima volta la tua Virtual Machine

Ti verrà chiesto di scegliere la ISO[1] di Debian 9. Se ancora non l'avessi fatto puoi scaricarla dal sito ufficiale[2]. Avvia l'installazione e procedi con essa.

È la tua prima volta che installi GNU/Linux? Niente paura, puoi scaricare l'ebook gratuito dal nostro sito (https://hacklog.net/it/hacklog-volume-1-anonimato/guida-a-debian) e seguirla passo-passo per avere un ambiente funzionante già dopo pochi minuti!

2.2 Ambiente di Difesa e WHLAB

Ok abbiamo la macchina per attaccare. Ma cosa attacchiamo, esattamente?

Escludiamo a priori attacchi a danni di siti web pubblici: oltre ad arrecare (inutili) danni ad altre aziende o persone, molto di ciò che verrà trattato in questo documento è consentito esclusivamente su ambienti di proprietà. Ricordiamoci infatti che l'abuso informatico è un reato penale e come tale bisognerebbe lavorare solo su ambienti controllati!

---

[1] Formato file che contiene un disco digitalizzato

[2] https://www.debian.org/distrib

Dunque dobbiamo creare un ambiente di lavoro; puoi scegliere una delle due strade proposte:

- Creare una Virtual Machine
- Ottenere l'accesso a WHLAB

2.2.1 VM o WHLAB?

WHLAB non è ancora attivo e verrà rilasciato solo quando sarà rilasciato Hacklog2. Info su www.hacklog.net

Personalmente mi sento di consigliare la Virtual Machine in quanto ci darà la possibilità di avere un ambiente di test più controllabile: in questo caso infatti potremo anche eseguire fix di sicurezza e verificare che queste siano in grado di mitigare gli attacchi. Grazie alla Virtual Machine avremo un vero e proprio server funzionante e in grado di rispondere agli attacchi in locale.

WHLAB è invece un ambiente di test, fornito da hacklog.net (https://hacklog.net/it/hacklog-volume-2/hacklog-whlab), che permette di interagire solo con un ambiente volutamente vulnerabile, senza offrire la possibilità di metterlo in sicurezza; inoltre è un progetto volatile (al momento non possiamo garantirne il funzionamento H24) e gli ambienti vengono resettati automaticamente, costringendoti a dover rigenerare nuovamente una nuova macchina. In compenso, ottenere un ambiente WHLAB è semplicissimo e richiede pochi click.

Di seguito una tabella con cui puoi fare le tue considerazioni e valutare con cosa iniziare:

	Virtual Machine	WHLAB
Facilità d'Uso	Variabile	Estremamente semplice
Completezza	Estremamente completa	Limitata
Reperibilità	In ogni momento	Limitata

	Virtual Machine	WHLAB
Costo	Free	Free (con limitazioni)
Tempi di setup	15 minuti - 2 ore (variabile)	5 minuti

2.2.2 Creare la Virtual Machine di Difesa

Utilizzare una Virtual Machine ci permetterà di mettere in pratica le fix di difesa che verranno proposte.

La procedura di creazione è identica a quanto visto nel capitolo X, così come il Sistema Operativo che andremo ad installare: in questo caso cambierà solo il nome della macchina che chiameremo *Hacklog 2 Difesa*.

Alla fine avrai due Virtual Machine (Hacklog 2 Attacco e Hacklog 2 Difesa) perfettamente funzionanti e pronte per essere configurate.

2.2.3 Configura la Virtual Machine di Difesa

In aggiunta ai pacchetti di default sarà necessario installare ulteriori applicazioni per rendere la macchina di difesa completa per questo corso. Se non hai intenzione di affrontare il capitolo Fondamentali del Web (capitolo X) dove verranno spiegati alcuni concetti fondamentali sui server dovrai installare anche:

- Apache (capitolo X)
- MySQL (capitolo X)
- phpMyAdmin (capitolo X)
- DVWA (capitolo X)

In caso di dubbi ti consigliamo di seguire il libro dall'inizio alla fine così da avere meno dubbi a riguardo.

2.3 Il Terminale

Se hai già letto l'*Hacklog, Volume 1: Anonimato* avrai già avuto modo di comprendere la necessità di utilizzare il Terminale e le infinite possibilità che quest'ultimo può offrirci.

Se invece è la prima volta che ne senti parlare allora dovrai da subito abituarti al suo utilizzo. Esso si presenta come un programma all'interno del nostro Sistema Operativo e consentirà di lanciare comandi e parametri necessari per raggiungere diversi scopi.

Nel libro il terminale verrà mostrato in questo modo:

```
$ ping www.inforge.net
```

Per motivi di comodità, utilizzeremo il simbolo $ (dollaro) per indicare il punto in cui lanceremo il comando indicato; considera che in certe situazioni nel tuo terminale potresti non trovare il simbolo ma il nickname e il nome macchina che stai utilizzando (es: stefano9lli@hacklog:).

Nell'esempio precedente il comando ping è il programma che richiamiamo mentre www.inforge.net è il parametro che passiamo a ping.

Troverai una lista di comandi utili da lanciare da Terminale al capitolo X.

2.4 Due VM in un'unica rete

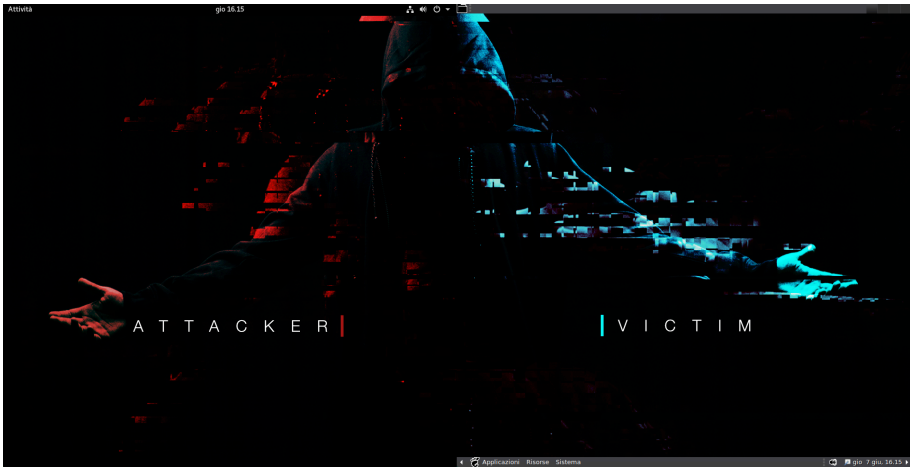



Figura X: rappresentazione dell'ambiente di lavoro. A sinistra la macchina d'attacco (riconoscibile dalla barra alta di GNOME3 e la scritta "ATTACCO") e la macchina di difesa (riconoscibile dalla barra in fondo di MATE e la scritta "DIFESA").

Per poter ritenere conclusa questa fase preliminare è necessario che entrambe le macchine siano collegate tra di loro e in grado di comunicare attraverso una rete (virtualizzata). In questo modo simuleremo a tutti gli effetti una condizione client-server (dove il client e la macchina che attacca e il server quella che subisce).

Dalle Impostazioni di ogni macchina virtuale (click destro su di essa, quindi Rete) configuriamo una nuova scheda di rete come segue:

- Clicca su "Scheda 2" quindi "Abilita scheda di rete"
- Connessa a: Rete interna
- Nome: hacknet
- In avanzate
  - Tipo di scheda Intel PRO/1000 MT Desktop
  - Modalità promiscua: Permetti tutto
  - Cavo connesso (abilitato)

Di seguito una screen di come dovresti avere le macchine affinché siano in grado di comunicare tra di loro:

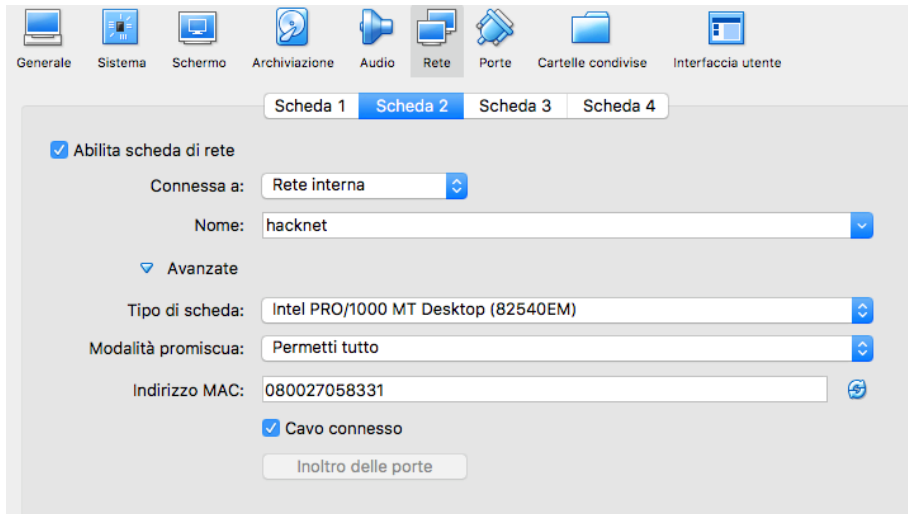



Figura X: esempio di configurazione della seconda scheda di rete delle due Virtual Machine

Potrebbe essere più comodo avere anche due indirizzi IP statici: ciò significa che, all'avvio, i due ambienti avranno sempre gli stessi indirizzi IP (statici) e non cambieranno in base alla presenza di altre VM o all'ordine di entrambe.

Possiamo dunque configurare uno speciale file all'interno dei due ambienti con il comando da terminale[1]:

```
$ ip a
```

Prendiamo appunti sui due identificatori delle schede di rete (nel nostro caso *enp0s3* e *enp0s8*) come da Figura X:

[1] Una breve spiegazione del terminale è a capitolo X

```
root@hacklog:/home/stefano9lli# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t qlen 1
    link/loopback 00:00:00:00:00:00 [illegible] 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host [illegible]
       valid_lft forever [illegible]erred_lft forever
    inet6 ::1/128 sc[illegible] host
       [illegible]orever preferred_lft forever
2: enp0s3: [illegible]CAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
[illegible]t qlen 1000
    link/ether 08:00:27:38:1d:78 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe38:1d78/64 scope link
       [illegible]_lft forever preferred_lft forever
3: enp0s8: [illegible]CAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
[illegible]1000
    link/ether 08:00:2[illegible]:83:31 brd ff:ff:ff:ff:ff:ff
    inet 20.0.0.2/24 brd 20.0.[illegible] scope global enp0s8
       valid_lft forever preferred_l[illegible]rever
    inet6 fe80::a00:27ff:fe05:8331/64 scope link
       valid_lft forever preferred_lft forever
```

Figura X: lanciando il comando "ip a" mostreremo gli identificatori delle schede Ethernet (virtuali).

Sempre da terminale, otteniamo i privilegi di root (su) e modifichiamo il file /etc/network/interfaces:

```
$ su
$ nano /etc/network/interfaces
```

e modifichiamolo sulla macchina **attacker** come segue:

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# La prima interfaccia di rete, collegata a Internet
auto enp0s3
iface enp0s3 inet dhcp
```

```
# Seconda interfaccia di rete, configurata in modo statico
auto enp0s8
iface enp0s8 inet static
address 20.0.0.2
netmask 255.255.255.0
gateway 20.0.0.1
broadcast 20.0.0.0
```

A questo punto puoi riavviare il servizio di networking (o meglio ancora riavviare completamente la macchina virtuale):

```
$ service networking restart
```

Ora dovrai effettuare gli stessi passaggi appena spiegati, cambiando nella seconda interfaccia l'address (indirizzo IP) della macchina **vittima**:

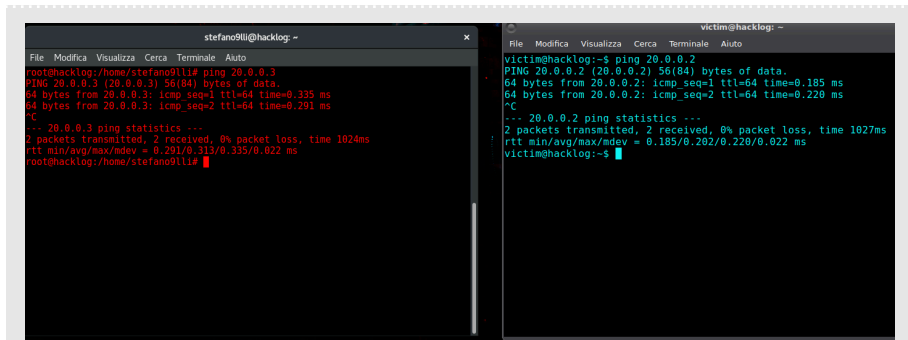
```
# Seconda interfaccia di rete, configurata in modo statico
auto enp0s8
iface enp0s8 inet static
address 20.0.0.3
netmask 255.255.255.0
gateway 20.0.0.1
broadcast 20.0.0.0
```

In questo modo avrai:

IP Attacker	IP Vittima
20.0.0.2	20.0.0.3

- IP attacker: 20.0.0.2
- IP vittima: 20..0.0.3

Per verificare la connettività tra i due, lancia sulle due macchine il comando:

```
$ ping <indirizzo_ip_dell_altra_macchina>
```

Quindi dalla macchina **attacker** digiterai:

```
$ ping 20.0.0.3
```

e dalla macchina **vittima** digiterai:

```
$ ping 20.0.0.2
```

Il risultato sarà una serie di risposte continue da entrambe le macchine (figura X):

Figura X: lanciando il comando ping effettuerai connessioni di test all'altra macchina. Puoi fermare il processo con la combinazione di tasti CTRL+C

In più potremmo voler definire un hostname così da non dover ogni volta indicare l'indirizzo IP della macchina vittima: da **attacker** modifichiamo il file /etc/hosts:

```
$ nano /etc/hosts
```

e aggiungiamo sotto gli altri indirizzi IP (figura X):

```
20.0.0.3    victim
```

Figura X: aggiungiamo l'hosts, così quando ci connetteremo ad esso riceveremo comunque risposta dal server

Salviamo con CTRL+X, quindi tasto S e INVIO (ricorda questa procedura, la useremo spesso nel corso della lettura). Puoi ora riprovare a "pingare" il nuovo host (ping victim) e dovresti poter comunicare facilmente con la macchina.

2.5 Ready, Set...

Alcune considerazioni prima di iniziare:

- Si prevede che l'utente abbia accesso diretto ad Internet
- Il testo da per scontato che la versione Debian in uso sia già aggiornata
- Alcuni link potrebbero essere non raggiungibili o oscurati; consigliamo l'utente di cercare le alternative adeguate tramite Motore di Ricerca
- Verrà fatto ampio uso di GIT: riteniamo che questo sia il modo più veloce e comodo per accedere a una vasta raccolta di applicazioni aggiornate. L'utente potrà comunque decidere di usare repository alternative o compilarseli in automatico.

3. I FONDAMENTALI DEL WWW

Se stai muovendo i tuoi primi passi nel mondo della Sicurezza Informatica e vuoi affrontare il World Wide Web è indispensabile che tu conosca come funziona a grandi linee il mondo di Internet. Cercherò di farla il più semplice possibile!

Troviamogli un significato: Internet è un insieme di sistemi connessi tra di loro che si scambiano messaggi. Alcuni possono accettare solo certi tipi di messaggi mentre altri solo da alcuni mittenti; una volta che queste informazioni vengono ricevute dal destinatario, quest'ultimo si occuperà di elaborarle.

Per evitare che ogni produttore software o hardware decida di fare di testa propria sono stati ideati gli RFC (Requests for Comments), documenti che stabiliscono gli standard in ambito informatico.

Prendiamo ad esempio l'HTTP: esso è un protocollo[1] che stabilisce in che modo un Browser Web e un Web Server comunicano tra di loro. Poiché sia il Browser che il Web Server sono d'accordo su come utilizzare questo protocollo (definito dai relativi RFC) allora sono in grado di comunicare.

Prima che tu ti chieda cosa sia un Web Server ti anticipo che ne parleremo a breve: per adesso ti serve sapere solo che è un programma, installato sul server, che si occupa di far caricare un sito web.

3.1 Cosa succede quando navighiamo?

In questo capitolo prenderemo confidenza con alcuni strumenti che usano i professionisti per determinare il traffico della rete; inoltre scopriremo come destrutturare il traffico di rete, riuscendo così ad analizzare i vari componenti e spiegarli dettagliatamente uno ad uno.

---

[1] Insieme di regole che consentono lo scambio di dati tra più dispositivi.

Ok, cosa succede quando navighiamo nel WWW?

1) Apriamo il Browser e digitiamo nella barra degli indirizzi **URL** qualcosa tipo: www.hacklog.net

2) Il tuo computer si connette ad un DNS: questo è un sistema che si occupa di tradurre il **dominio** (www.hacklog.net) nel relativo **Indirizzo IP** (in questo caso 104.28.5.97): una specie di elenco telefonico per intenderci.

3) Il tuo computer cercherà ora di stabilire una connessione di tipo TCP[1] sulla porta[2] 80 oppure 8080, solitamente usate per il **traffico HTTP** (nella versione in HTTPS si collegherà invece alla porta 443).

4) Se la connessione avrà successo il tuo Browser invierà al Web Server una **richiesta HTTP** come la seguente:

```
GET / HTTP/1.1
Host: www.hacklog.net
Connection: keep-alive
Accept: application/html, */*
```

5) Il server, se avrà capito la tua richiesta e la accetterà, **risponderà** con qualcosa del genere:

```
HTTP/1.1\ 200 OK
Content-Type: text/html
<html>
    <head>
          <title>Benvenuto su Hacklog.it!</title>
    </head>
    <body>…</body>
</html>
```

[1] Protocollo da cui HTTP eredita alcune logiche di trasmissione.

[2] Nelle reti informatiche, una porta è uno strumento virtuale che consente di riservare un tipo di connessione dalle altre, così da permettere più connessioni contemporaneamente. Ogni indirizzo IP può avere massimo 65.535 porte.

6) Il tuo Browser raccoglierà la risposta ed effettuerà il **render**[1] del codice ottenuto. In questo caso sarà la homepage del sito www.hacklog.net

Facile, non è vero? Beh, ce n'è ancora di strada da fare :)

3.2 La dura vita del Web Server

L'esempio che abbiamo visto prima ritorna la pagina principale di un sito: questo succede perché, risolvendo l'host principale (nel caso precedente www.hacklog.it), viene caricata la pagina di default che il web server ha scelto di mostrare. In un web server base solitamente la pagina principale è il file **index.html** .

Il compito del Web Server è quello di assicurarsi che, data una certa *richiesta HTTP*, venga fornita la corretta *risposta HTTP*; il risultato di questo rapporto è l'output a video che siamo soliti vedere nella schermata di un Browser (appunto la pagina web).

Esistono diversi Web Server, tra i più comuni citiamo: *Apache* (il più popolare HTTP Server o la versione Tomcat), *Nginx*, *Lighttpd*, *ColdFusion*, *Mongoose*, *IIS, Cherokee* e così via.

Ogni Web Server ha le proprie particolarità e peculiarità: laddove infatti IIS è preferibile su delle macchine Windows, allo stesso modo si preferiscono Apache/Nginx/Lighttpd su Sistemi Operativi a base UNIX, tuttavia una non esclude l'altra. La scelta di un Web Server è comunque a discrezione dell'amministratore di sistema.

Tornando a noi, per capire meglio in che modo un web server si comporta sarebbe opportuno installarne uno! Perché non provarci subito?

Sulla macchina **vittima** apri il Terminale e digita:

```
$ su
$ apt install apache2 apache2-doc
```

---

[1] Restituire graficamente a video

Riassumendo:

- con **su** abbiamo elevato i permessi del nostro utente a root (verrà quindi chiesta la password che abbiamo scelto in fase di installazione)
- con **apt install** installeremo il pacchetto di **apache2** (il Web Server) e la relative documentazione.

Apriamo il Browser dalla macchina **attaccante** e visitiamo l'indirizzo: http://victim . Se tutto va per il verso giusto dovremmo ottenere la schermata iniziare di Apache2:

Figura X: colleghiamoci all'IP della macchina vittima e caricheremo la pagina di default di Apache

Qualora dovessero esserci problemi prova a verificare lo status del Web Server:

```
$ service apache2 status
```

Dovremmo ricevere un messaggio del genere:

```
apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset:
   Active: active (running) since Tue 2018-02-27 16:41:39 CET; 3s ago
...
```

Alla voce Active dovrebbe comparire lo status "active (running)". Se così non fosse riavviamo i servizi con:

```
$ service apache2 restart
```

sostituendo "restart" con "stop" o "start" per comandare le medesime azioni.

Quello che abbiamo appena fatto è installare uno dei tanti Web Server presenti nella rete: per convenzione, useremo Apache2 in quanto è ancora oggi uno dei più popolari in circolazione (e di conseguenza uno dei più documentati).

Prima di andare avanti ci sarà utile effettuare una piccolissima modifica al web server: in sostanza dovremo abilitare la lettura degli .htaccess, dei file speciali (nascosti) che ci consentono di creare configurazioni in modo veloce ed eseguire tutti i nostri test. Dalla macchina vittima modifichiamo il file di configurazione Apache:

```
$ nano /etc/apache2/apache2.conf
```

cerchiamo la porzione seguente (usa CTRL+W per ricercare tramite nano):

```
<Directory /var/www/>
        Options Indexes FollowSymLinks
        AllowOverride None
        Require all granted
</Directory>
```

e assicuriamoci che il valore "AllowOverride" sia su All [figura x]. Chiudi con CTRL+X, S e INVIO.

```
  GNU nano 2.7.4          File: /etc/apache2/apache2.conf


<Directory /usr/share>
        AllowOverride None
        Require all granted
</Directory>

<Directory /var/www/>
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
</Directory>

#<Directory /srv/>
#       Options Indexes FollowSymLinks
#       AllowOverride None
#       Require all granted
#</Directory>
```

Figura X: consentiamo agli .htaccess di usare le proprie configurazioni

Una volta fatto ciò, riavvia ancora una volta i servizi di Apache2.

```
$ service apache2 restart
```

3.2.1 Hosting, Cloud, VPS e Server

A dirla tutta non c'è bisogno di essere degli amministratori di sistema per caricare un'applicazione web in rete, perlomeno non più: negli anni il mercato si è interessato alla fornitura di servizi pensati esclusivamente per il web, offrendo ai loro utenti lo spazio necessario a far funzionare qualunque applicativo lasciando che dei team di esperti si occupino dell'amministrazione.

Sono quindi sempre più popolari i servizi di **hosting**, spazi web in cui è possibile trovare già pre-installati tutti i programmi necessari: ne esistono anche di gratuiti, relativamente stabili e comodi per progetti amatoriali e no-profit.

Inoltre da qualche anno a questa parte si sente sempre più spesso parlare di **cloud**: sulla scia dell'hosting, il mercato cloud risponde a quelle esigenze che sempre più spesso un web master si ritrova ad affrontare nel tempo (performance limitate, servizi scadenti etc...). Nei sistemi cloud odierni l'utente esternalizza i propri servizi, ad esempio i server di posta, i database, le risorse statiche (immagini, video etc...) in macchine specificatamente ottimizzate, pagando solo il consumo che ne fa (o che ne fanno i loro utenti tramite il sito web).

Se invece il web master ne ha le competenze può optare per le **VPS** (Virtual Private Server): immagina di avere un grande server e di spezzarlo in piccole parti. Ogni macchina è virtualizzata ed è dimensionata in base alle richieste del web master, così da avere un Sistema adeguato ai suoi scopi senza esser costretto a spendere un occhio della testa.

Quando il gioco si fa duro allora il web master avrà bisogno di un **Server** dedicato a tutti gli effetti: i costi sono variabili e possono oscillare da 50€ fino a qualche decina di migliaia di € (dipende ovviamente dalle risorse che si pretendono). Queste sono spesso macchine in grado di sostenere grandi portate di traffico.

È possibile inoltre che sia VPS che Server siano fornite con formula **managed**: in questo caso un team di sistemisti si occupa di gestire la macchina al posto vostro, offrendovi la libertà di modificare a piacimento le risorse ma mantenendo la semplicità di un hosting; le formule managed sono ovviamente molto più costose e possono incidere dai 100/150€ al mese a salire per ogni macchina.

3.2.2 Reverse Proxy Server

Con l'arrivo di nuovi dispositivi, reti, normative sulla privacy e la necessità di offrire siti web sempre più veloci, i servizi di **Reverse Proxy Server** e molti altri hanno visto incrementare la loro rete di clienti in modo esponenziale, diventando elementi praticamente indispensabili per i webmasters di tutto il mondo.

Un reverse proxy server si pone tra la risoluzione di un host (ad esempio www.hacklog.net) e l'indirizzo IP (Figura X); può avere diversi scopi e offrire vantaggi di diverso tipo[1]:

- **Geolocalizzare la risposta**: è in grado di rispondere più velocemente in quanto offre una CDN distribuita geograficamente sul territorio. Alla richiesta di un utente, sarà il server più vicino in termini di distanza a rispondere.

---

[1] Ogni fornitore di Reverse Proxy Server può decidere di offrire (gratuitamente o a pagamento) o meno queste caratteristiche. Il testo vuole solo illustrare in che modo si possono applicare i Reverse Proxy Server di fronte alla risoluzione di un IP.

- **Velocità al portale**: sono spesso forniti tool in grado di ottimizzare automaticamente le immagini, offrire le ultime versioni del protocollo HTTP, comprimere le risorse presenti sul portale, stabilire regole di cache e molto altro.
- **Stabilità**: possono offrire copie cache dei portali qualora quest'ultimi vadano offline e distribuire il carico di lavoro attraverso un processo definito load balancing
- **Sicurezza**: sono spesso presenti integrazioni automatiche dei certificati SSL/TLS per offrire protocolli di tipo HTTPS, Firewall, IDS[1], blocco automatico di bot malevoli, protezione ad attacchi DDoS[2] e così via.

Esistono diversi fornitori di questi servizi, tra i più popolari citiamo: *Cloudflare, Sucuri, Incapsula, Google Project Shield, Akamai* e molti altri.

Come vedremo nel capitolo X, questi possono essere un ottimo deterrente (purché ben configurati) per mitigare attacchi di vario genere.

3.2.3 Dal Dominio all'IP (DNS)

Acronimo di DNS, i **Domain Name System** sono dei sistemi che consentono di convertire il nome di un dominio (ad esempio: hacklog.it) in un indirizzo IP: proprio come un elenco telefonico, è molto più semplice ricordare per un essere umano il nome di un sito piuttosto che una serie di numeri.

Questo processo può aver luogo nella cache locale – solitamente di un computer o di un router/modem – oppure nel cosiddetto "zone file" di un server, un file che contiene le informazioni dei DNS rappresentati dai Resource Records (RR); parleremo di questi a breve, prima cerchiamo di capire come funziona un DNS.

---

[1] Strumenti, software o hardware, in grado di identificare il tentativo di intrusione a un sistema informatico.

[2] Attacchi a negazione di servizio, metodi efficaci per saturare le risorse di un sistema causandone il blocco.

Immaginiamo di aprire il sito hacklog.it: il Sistema Operativo si occupa di verificare se questo è stato già caricato precedentemente, quindi per prima cosa ne verificherà la presenza nella cache locale.

All'interno della cache sono scritti tutti gli indirizzi IP con i rispettivi host: questo permette di evitare che il Sistema Operativo interpelli la rete Internet ogni volta che deve navigare in rete; se questo non è presente allora interpellerà un DNS Server, un server spesso fornito dall'ISP[1] oppure esterno (vedesi Google, OpenDNS etc...).

3.2.3.1 RISOLUZIONE BASE DI UN DNS

Tutto il processo ovviamente è "trasparente" agli occhi dell'utente ma è possibile mettere a nudo una richiesta DNS attraverso uno dei tanti strumenti già preinstallati nelle distribuzioni GNU/Linux.

Per fare alcuni test avremo bisogno di dnsutils, un pacchetto che contiene alcuni tools per testare la rete. Da terminale lanciamo:

```
$ apt install dnsutils
```

Vediamo un facile esempio come segue (lancia il comando nslookup da Terminale, quindi compila ogni riga come segue, per chiudere il programma CTRL+C):

```
$ nslookup
> set type=A
> hacklog.net
```

La risposta sarà la seguente:

```
Server:     192.168.0.1
Address:    192.168.0.1#53

Non-authoritative answer:
Name:       hacklog.net
Address: 104.27.162.41
```

[1] Internet Service Provider, più comunemente il "provider Internet"

```
Name:      hacklog.net
Address: 104.27.163.41
```

Cerchiamo di capire in cosa consistono questi comandi e il risultato ottenuto:

1) Nella prima riga abbiamo specificato "set type=A". Questo significa che stiamo richiedendo tutti i record[1] di tipo A; spiegheremo a breve di che si tratta

2) Nella seconda riga specifichiamo il dominio/host

3) La terza e quarta linea rappresentano gli indirizzi IP del server DNS interpellato. Noterai come nella quarta linea viene indicato #53: questo rappresenta il numero della porta utilizzata per la richiesta che abbiamo fatto. Di default, i server DNS rispondono sulla porta UDP 53.

4) In fondo vengono mostrate le risposte non-autorevoli (Non-authoritative answer): questo significa che il nostro server DNS sta usando a sua volta un altro server DNS da cui attinge per risolvere la richiesta.

3.2.3.2 TIPI DI RECORD

Un Zone file è rappresentato da un semplice file di testo: all'interno di esso sono racchiusi i Resource Records, delle righe che stabiliscono in che modo ogni singolo record risolve l'indirizzo IP.

Per capire in cosa consiste un Record prendiamo in esempio la tabella X:

Nome	Tiplogia	Valore
hacklog.net	A	104.27.163.41
mail	A	104.27.163.41
www	CNAME	alias di hacklog.net
hacklog.net	MX	handled by hacklog.net

L'esempio dimostra come è strutturato un semplicissimo zone file con all'interno 3 Resource Records: questo ci permette di capire il motivo per cui abbiamo specificato la tipologia "any" al capitolo X.

[1] I risultati di una risoluzione DNS

Possiamo quindi verificarne il funzionamento utilizzando una nuova tipologia, ad esempio MX (record di posta, Mail eXchange):

```
$ nslookup
> set type=MX
> hacklog.net
```

con il seguente risultato:

```
Server:    192.168.0.1
Address:   192.168.0.1#53

Non-authoritative answer:
hacklog.net      mail exchanger = 5 ***.
hacklog.net      mail exchanger = 1 ***.
hacklog.net      mail exchanger = 100 ***.
```

Il risultato ottenuto farà riferimento non più alla risoluzione del dominio come se fosse una pagina web ma piuttosto al "manager di posta".

In certi casi sono i record TXT quelli in grado di offrire molte informazioni; ad esempio lanciando:

```
$ nslookup
> set type=TXT
> hacklog.net
```

si può sapere che il dominio hacklog.net fa uso anche di un servizio di posta esterno (m*****t) per inviare email (potrà essere utile in fase di OSINT, capitolo X):

```
hacklog.net      text = "v=spf1 include:spf.mailjet.com
include:mx.ovh.com ~all"
hacklog.net      text = "1|www.hacklog.net"
```

Esistono diverse tipologie di Record, citiamo i cinque più popolari:

- **Record A**: mappa un indirizzo IP a un hostname (es: 104.27.163.41 a hacklog.net)
- **Record MX**: reindirizza le email al server che ne ospita gli account

- **Record NS**: definisce i server che comunicherà le informazioni DNS relative a un dominio
- **Record CNAME**: definisce gli alias verso il nome di dominio reale (es: www.hacklog.net riporterà a hacklog.net)
- **Record TXT**: fornisce informazioni di testo e possono essere utili per vari scopi (ad esempio confermare che si è i proprietari di un sito)

3.3 Hello, World!

Torniamo alla nostra pagina iniziale di Apache2. Dov'è contenuta? Dove si modifica? Come si creano altre pagine?

Prima di rispondere a queste domande proviamo a lanciare da macchina **vittima** il seguente comando:

```
$ nano /var/www/html/index.html
```

Come probabilmente ricorderai usiamo *nano* come editor di testo per crea e modificare i file direttamente da Terminale. Segue poi il percorso del file: in questo caso Apache2 recupera tutti i file presenti nella cartella /var/www/html. Qui potrai caricare tutte le pagine che vorrai ed evocarle direttamente da Browser.

Non è detto che sul sistema attaccato il cyber-criminale si trovi sempre pre-installato nano, né i permessi per installarlo. Talvolta l'editor consigliato è **vi** o la variante **vim**, tuttavia per motivi di semplicità eviteremo di utilizzarlo.

Tornando all'editor, ci verrà mostrata una schermata contenente del codice HTML (su cui torneremo). Questa è la versione nuda e cruda della pagina demo di Apache2 vista in precedenza. Per il momento ignoriamone il contenuto, quindi usciamo da nano (ricorda le combinazioni CTRL+X, se hai difficoltà c'è sempre il cheatsheet al capitolo X). Rinominiamo il file appena aperto:

```
$ mv /var/www/html/index.html /var/www/html/
index.backup.html
```

In questo modo il file index.html verrà rinominato in index.backup.html . Certo che, con tutti questi path inizia ad essere difficile lavorare, quindi possiamo spostarci direttamente nella cartella:

```
$ cd /var/www/html
```

Da questo momento in poi non sarà più necessario specificare il percorso di destinazione in quanto siamo già presenti in essa. Per esserne tuttavia sicuri lanciamo il comando:

```
$ pwd
```

Il sistema ci risponderà quindi con:

```
/var/www/html
```

Possiamo ora riaprire il nostro editor di testo preferito e creare un nuovo file index.html:

```
$ nano index.html
```

Buttiamoci dentro il nostro testo di prova come segue:

```
Hello, World!
```

Salva il file con [CTRL+X], [S] per confermare e [INVIO] per eseguire le modifiche.

Apriamo ora il browser e navighiamo all'indirizzo: http://victim e vedrai comparire il messaggio appena creato. Se decidessi di visitare http://victim/index.html noterai che verrà caricata la stessa pagina, proprio per il principio della pagina index.html come la prima ad essere caricata.

E se volessimo creare una nuova pagina? Basta ricreare il processo, questa volta creando un file con un nuovo nome.

```
$ nano hacklog.html
```

Scriviamo qualunque cosa, quindi salviamo il file e carichiamo sul browser: http://localhost/hacklog.html

Per renderti la vita più semplice immagina che localhost sia il sinonimo di /var/www/html, quindi:

Path	URL
/var/www/html/index.html	http://victim/index.html
/var/www/html/hacklog.html	http://victim/hacklog.html
/var/www/html/test.html	http://victim/test.html

e via dicendo.

Se dovessi avere qualche dubbio su quali file sono presenti nella cartella in cui ti trovi ti ricordiamo il comando che permette di ottenere una lista di file e cartelle presenti nel path in cui ti trovi:

```
$ ls
```

Se hai seguito alla lettera i comandi dovresti quindi ottenere il seguente output:

```
hacklog.html index.backup.html  index.html
```

Perdersi nella navigazione da Terminale è facile. Ricorda che puoi usare il comando cd per spostarti tra le varie cartelle. Se non dovessi ricordare come si fa ti consigliamo di aprire il capitolo X.

All'inizio potrà sembrare ostico e a tratti "strano" navigare da Terminale: tuttavia se vuoi immedesimarti come un cyber-criminale dovrai essere in grado di utilizzare a menadito la shell testuale piuttosto che quella grafica. Capirai più avanti il perché di ciò!

Nota avanzata: come probabilmente avrai notato abbiamo lasciato i permessi all'utente root. Questa pratica non è assolutamente consigliata per un sistema di produzione in live e pertanto si consiglia di abilitare il modulo userdir e di assegnare permessi e cartelle come da procedura standard. Maggiori informazioni al link: https://wiki.debian.org/it/LaMp

3.3.1 HTML, le fondamenta del Web

Per progettare efficacemente il Front-end, la parte ciò visibile solo all'utente (spesso chiamata GUI, Graphic User Interface), nei primi anni '90 è stato ideato un linguaggio chiamato HTML.

L'HTML (HyperText Markup Language) viene spesso erroneamente associato ad un linguaggio di programmazione ma, come dice la parola stessa, la definizione più corretta è **linguaggio di mark-up**.

Come linguaggio, infatti, ha poco a che vedere con funzioni, condizioni, variabili e via dicendo: la sua funzione è limitata allo strutturare le fondamenta di un'applicazione web.

Vedi insomma l'HTML come il progetto di architettura di una pagina web: ti permette di inserire pulsanti, tabelle, contenitori, link etc… consentendo anche una limitata formattazione grafica per abbellire, entro certi limiti, le pagine.

Allo stato attuale ha raggiunto la versione 5, quindi lo troverai chiamato spesso anche HTML5: le pagine create con alla base solo HTML vengono dette **statiche** e sono salvate in formato **.htm** e **.html** .

Il linguaggio HTML si presenta in questo modo:

```
<html charset="UTF-8">
    <head>
        <!-- Questo è un commento -->
        <title>Benvenuti su hacklog.it!</title>
    </head>
    <body>
        <h1>Questa è una pagina di prova :)</h1>
        <div>Qui c'è del testo</div>
    </body>
</html>
```

Perché non proviamo a copia-incollare questa porzione di codice nel neonato index.html nella macchina **vittima**? Il risultato sarà come in figura X.

Figura X: una semplice pagina HTML a cui siamo collegati sulla macchina vittima.

Come dicevo questo non sarà un corso di programmazione, tuttavia lasciare l'utente in balia di un vicolo cieco mi sembra poco professionale. Diamo allora un significato a quel poco che abbiamo scritto:

- **L'HTML si basa sulle tag**: le tag si aprono e si chiudono con i simboli di maggiore e minore. Per convenzione (quasi) tutte le tag si aprono e si chiudono e possono racchiudere in esse altre tag.

- **L'HTML si può commentare**: tutto ciò che è presente tra <!-- e --> non viene considerato dal browser ma può essere letto nel codice sorgente[1]. Vedremo successivamente come analizzare quest'ultimo.

- **L'HTML si indenta**: come vedi ci sono degli spazi (assegnati con il tasto [TAB] della tastiera). Sebbene non strettamente necessario, è una convenzione utilizzata dai programmatori per leggere meglio il codice e capirne la struttura gerarchica.

Spendiamo due parole anche sulle tre tag fondamentali dell'HTML:

---

[1] Il codice che compone un programma.

- **<html>**: All'interno di questa tag si comunica al browser che è presente del codice HTML
- **<head>**: All'interno di questa tag si comunica al browser dei parametri che il browser non mostra all'utente ma che può utilizzare per il corretto funzionamento di una pagina web
- **<body>**: All'interno di questa tag si comunica al browser che il contenuto va mostrato all'utente

---

Per creare una pagina HTML basta un **editor di testo** (sebbene esistano IDE e WYSIWYG per facilitarne lo sviluppo) e non ha bisogno di un Web Server per funzionare.

Tieni a mente che esistono centinaia di tag HTML pensate per i diversi scopi: un buon punto di partenza per conoscerli davvero tutti è all'indirizzo https://www.w3schools.com/html/ . Continuando con il testo vedremo comunque l'utilizzo di alcuni tag fondamentali per la progettazione di un'applicazione web.

3.3.2 CSS, la "mano di vernice"

Laddove il linguaggio HTML manca di "pennelli" e "righelli" subentra un nuovo linguaggio, anch'esso non di programmazione ma orientato alla **formattazione dei documenti**.

Se l'HTML è la struttura portante, il CSS è la mano di vernice: permette di definire uno stile ai contenuti, colorando ad esempio i contenitori, scegliendo i font che la pagina dovrà usare, dimensionare le tabelle e così via.

Il linguaggio CSS dipende sempre da una pagina HTML e può essere incluso direttamente nel codice HTML:

```
<html>
    <head>
        <!-- Questo è un commento -->
        <title>Benvenuti su hacklog.it!</title>
```

```
        <meta charset="UTF-8">
        <style type="text/css">
            /* Questo è un commento CSS */
            body
            {       background:yellow;    }
            h1
            {       text-decoration:underline;   }
            div
            {       color:red       }
        </style>
    </head>
    <body>
        <h1>Questa è una pagina di prova :)</h1>
        <div>Qui c'è del testo</div>
    </body>
</html>
```

Tuttavia sarà molto più probabile trovare un link a un foglio di stile esterno, questo per sfruttare tutte le caratteristiche di browser e web server per caricare parallelamente e in forma statica le risorse collegate.

Proviamo allora a creare un file con il comando:

```
$ nano style.css
```

che contiene il seguente contenuto:

```
body
{    background:yellow;    }
h1
{    text-decoration:underline;   }
div
{    color:red      }
```

Salviamo il file e riapriamo il nostro index.html:

```
$ nano index.html
```

e modifichiamolo come segue:

```
<html>
    <head>
        <!-- Questo è un commento -->
        <title>Benvenuti su hacklog.it!</title>
        <meta charset="UTF-8">
        <link rel="stylesheet" type="text/css"
href="style.css">
    </head>
    <body>
        <h1>Questa è una pagina di prova :)</h1>
        <div>Qui c'è del testo</div>
    </body>
</html>
```

Ricarichiamo la pagina http://victim e vedremo le modifiche prendere forma. Questa è Facciamo anche qui alcune considerazioni sul CSS:

- **Il CSS vive di selettori**: come hai visto abbiamo usato i tag body, h1 e div: questi elementi sono già presenti nell'HTML e permette di interfacciarsi con essi
- **Il CSS contiene proprietà e valori**: all'interno delle parentesi graffe { e } è possibile specificare una proprietà (un termine inglese) seguito da un valore. La traduzione del selettore *body* dice: lo sfondo (background) è giallo (yellow).
- **Il CSS si può commentare**: tutto ciò che è contenuto tra /* e */ non viene considerato dal Browser
- **Il CSS si indenta**: come per l'HTML si seguono delle convenzioni per comprendere meglio il linguaggio. Ne esistono diverse, ognuna che segue una linea di pensiero.

3.3.3 Javascript, il client tuttofare

Con l'evoluzione del World Wide Web gli sviluppatori hanno chiesto sempre più funzionalità per far interagire l'utente con i loro portali: nonostante abbiamo visto molto poco dei linguaggi di mark-up uno dei limiti che forse avrai notato è che non possono leggere le azioni di un utente.

Mi spiego: HTML e CSS generano solo degli output, non sono in grado di capire (nonostante siano stati fatti molti passi in merito) **cosa sta facendo un utente**: sta muovendo il mouse? Sta scrollando la pagina? Vogliamo dirgli qualcosa di nuovo senza che ricarichi la pagina? Ecco a cosa serve un linguaggio di clientscript: diamo il benvenuto a Javascript.

In una pagina web il Javascript si presenta in questo modo:

```
<html>
    <head>
        <!-- Questo è un commento -->
        <title>Benvenuti su hacklog.it!</title>
        <meta charset="UTF-8">
        <link rel="stylesheet" type="text/css"
        href="style.css">
        <script type="text/javascript">
            // Questo è un commento
            /* Anche questo lo è! */
            alert("Hello, World!");
        </script>
    </head>
    <body>
        <h1>Questa è una pagina di prova :)</h1>
        <div>Qui c'è del testo</div>
    </body>
</html>
```

Tuttavia, come per il CSS, si preferisce richiamarlo da un file esterno in formato **.js**:

```
<html>
    <head>
        <!-- Questo è un commento -->
        <title>Benvenuti su hacklog.it!</title>
        <meta charset="UTF-8">
```

```
            <link rel="stylesheet" type="text/css"
            href="style.css">
            <script type="text/javascript" src="script.js"></
script>
    </head>
    <body>
            <h1>Questa è una pagina di prova :)</h1>
            <div>Qui c'è del testo</div>
    </body>
</html>
```

Vogliamo provare? Allora copia-incolla la riga aggiunta nel tuo file index.html (ti ricordi come si modifica si?), quindi creiamo ancora un nuovo file:

```
$ nano script.js
```

e aggiungiamo il seguente codice:

```
alert("Hello, World!");
```

Caricando la pagina all'indirizzo http://victim otterremo la nostra pagina Web vista in precedenza, seguita dal popup con su scritto "Hello, World!". Abbiamo quindi evocato la funzione base alert che permette di mostrare a video ciò che vogliamo. Facciamo anche in questo caso le nostre considerazioni:

- **Javascript è un linguaggio di scripting**: a differenza dei primi due è un linguaggio a tutti gli effetti e richiede che il codice sia scritto correttamente (altrimenti potrebbe fare le bizze!)
- **Javascript è un linguaggio interpretato**: può usare quindi variabili, oggetti, array, condizioni, funzioni e tutto quello che si trova in un normale linguaggio di programmazione
- **Javascript si può commentare**: i commenti supportati sono su // una riga o su /* più righe */

Mi rendo conto che la spiegazione può risultare riduttiva ai molti, vedremo a tempo debito alcune caratteristiche fondamentali di questo e di altri linguaggi.

Ricorda: quando devi lavorare con le stringe (caratteri alfabetici, numeri e così via) devi metterle tra apici (') o virgolette ("); se lo dimentichi lo script andrà in errore!

Nota avanzata: nonostante il Javascript sia stato originariamente concepito per operare solo in lato client, esso può essere un ottimo strumento anche per comunicare con il back-end. Basti vedere le applicazioni possibili con AJAX o da qualche anno con il forte interesse della web community per Node.js .
Inoltre, molte applicazioni fanno uso di librerie e framework che ne potenziano enormemente le capacità: due esempi molto popolari sono indubbiamente jQuery e AngularJS.

Attenzione! Javascript NON è Java!

3.4 Navigare nel Web

Fino ad ora abbiamo visto come è possibile caricare pagine, senza però spostarsi da una pagina all'altra.

Questa funzione è prerogativa dell'HTML e in particolar modo del tag <a> che, attraverso l'attributo href, può creare un link per consentire all'utente di cambiare pagina.

Riapriamo quindi il nostro file index.html sulla macchina **vittima**:

```
$ nano index.html
```

e aggiungiamo prima della chiusura del body la seguente riga:

```
...
    <a href="hacklog.html">Vai alla pagina Hacklog</a>
    </body>
```

```
</html>
```

Ricarichiamo la pagina http://victim dal nostro browser e clicchiamo sul link. Verremo riportati alla seconda pagina che abbiamo creato. Facile vero?

Perché non provi a inserire un link nella pagina hacklog.html che riporta questa volta alla homepage? Tieni presente che nell'attributo href puoi indicare anche un URL, ad esempio: http://victim/index.html

3.4.1 URL

Come probabilmente saprai qualunque Web Browser funziona secondo una logica che consiste nel navigare attraverso *collegamenti ipertestuali*.

Questi collegamenti vengono chiamati URL (acronimo di Uniform Resource Locator), una sequenza di caratteri che identifica l percorso in rete di una determinata risorsa, che sia una pagina web oppure un video/immagine/audio: l'URL a cui ci si trova è presente nella *barra degli indirizzi* del Web Browser.

Gli standard[1] definiscono l'URL attraverso la seguente struttura:

```
protocollo://user:pass@host:port/path?query#frag
```

Ad esempio:

```
http://stefano:pwd@hacklog.it:21/var/www?log=true#debug
```

(Tutto ciò non ha alcun senso, quindi aprirlo non darà alcun risultato. È un esempio, prendilo come tale :P).

Cerchiamo di capire in breve di cosa si tratta:

- **protocollo –** è il primo elemento e identifica il tipo di protocollo in uso. Tra i protocolli di uso comune troviamo HTTP, HTTPS e FTP .
- **://** – separa il protocollo dal resto dell'URL (vedi capitolo successivo)

[1]Stabilito dal W3C, il consorzio che definisce gli standard del World Wide Web.

- **user:pass@** (opzionale) – permette di fornire credenziali d'accesso al web browser.
- **host** – è il nome del dominio o l'indirizzo IP che verrà interrogato per la navigazione.
- **port** (opzionale) – identifica la porta associata al servizio. Se non specificata sarà 80 per HTTP, 443 per HTTPS e 21 per FTP.
- **path** (opzionale) – identifica il percorso da raggiungere all'interno del server. Solitamente a ogni slash (/) aggiuntivo ci si riferisce ad una sottocartella aggiuntiva.
- **query-string** (opzionale) – permette di fornire ulteriori informazioni al server in modo che questi possa elaborarne il contenuto. La query-string inizia con un punto interrogativo (?) per poi seguire una variabile e il relativo valore (variabile=valore). Qualora siano presenti più elementi vengono concatenati con la e commerciale (&).
  Esempio: www.sito.it/index.php?animale=gatto&nome=pucci
- **frag** (opzionale) – indica una posizione all'interno della risorsa, solitamente un anchor link all'interno di una pagina web.

3.4.2 Il Protocollo

Prima di ritornare a parlare di sviluppo di un'applicazione web soffermiamoci sui protocolli, in quanto non avremo modo di rivederli con calma più in la.

Cerchiamo una definizione semplice: il protocollo di rete è quell'insieme di **regole** che determina in che modo due apparecchi informatici devono poter comunicare senza errori.

Prendendo un esempio non-informatico, il Protocollo di Kyoto è un trattato internazionale che stabilisce alcune regolamentazioni circa l'inquinamento atmosferico globale tra 180 paesi, i quali si differenziano per tipo di economia, geografia del territorio, lingua, religione e molte altre sfumature. Grazie al

protocollo, tutti gli Stati sono in grado di raggiungere l'obiettivo comune, nonostante le differenze culturali, territoriali e linguistiche.

Ogni protocollo ha delle qualità che lo rendono ideale in diverse situazioni[1], ad esempio:

- **Protocollo TCP**: è il protocollo su cui si basa il funzionamento della maggioranza delle applicazioni in rete e che consente un'affidabile scambio di informazioni tra un host e l'altro
- **Protocollo UDP**: è un protocollo meno affidabile del TCP ma più veloce; viene spesso utilizzato in situazioni dove è necessario trasmettere le informazioni in modo più rapido (ad esempio lo streaming o in un videogioco online)
- **Protocollo HTTP**: è il protocollo standard per l'interpretazione di informazioni all'interno del World Wide Web

I browser più blasonati sono in grado di gestire centinaia di protocolli e non è detto che tutti siano disponibili per tutte le piattaforme. Per facilitare l'esposizione possiamo dire che esistono quattro tipi di protocolli.

Protocolli gestiti dal browser

Questa tipologia di protocolli, come dice la parola stessa, viene gestita direttamente dai browser; quest'ultimi si occupano di mostrare a schermo le informazioni e offrirne l'interattività necessaria ai fini d'utilizzo.

Di questa categoria i protocolli più comuni sono **HTTP** e **HTTPS** per la navigazione web, mentre più raramente potremmo trovarci di fronte all'**FTP**, un tipo di protocollo utilizzato per il trasferimento di file.

Protocolli di terze parti

In questa tipologia ritroviamo protocolli gestiti da programmi esterni: ad esempio per avviare una chiamata Skype potremmo trovare il protocollo **skype://**, mentre

[1] Puoi trovare una lista più completa all'indirizzo: https://it.wikipedia.org/wiki/Protocollo_di_rete

Telegram fa uso di **tg://**; sicuramente uno dei più popolari - diventato a tutti gli effetti uno standard - è **mailto:** che permette di interagire con un software di posta per inviare una mail (è interessante notare come, in quest'ultimo, non sono presenti i due slash - // - dopo i due punti). Un esempio banale è il seguente:

```
mailto:info@hacklog.net
```

Concludiamo la lista con i protocolli interni dei browser: questi consentono di gestire le impostazioni interne dell'intero programma attraverso una propria interpretazione di utilizzo, fuori quindi da ogni standard. In Chrome ritroviamo quindi **chrome://** e in Firefox, ovviamente, **firefox://**

Protocolli non incapsulanti

Al giro di boa della nostra lista ritroviamo dei protocolli che permettono di interagire col motore interno del browser.

Tra questi citiamo **javascript:** che consente di generare codice lato client dell'omonimo linguaggio di scripting, e **data:** che invece rende possibile la creazione di piccoli documenti – volendo anche immagini – senza dover effettuare una nuova richiesta in rete.

In un qualunque browser potremmo quindi scrivere nella barra degli indirizzi:

```
javascript:alert("Ciao a tutti da Stefano Novelli!");
```

Per visualizzare una alert box attraverso una semplice funzione con il linguaggio Javascript.

```
Protocolli incapsulanti
```

L'ultima famiglia dei protocolli che andiamo a illustrare vengono utilizzati di fronte a risorse esterne, permettendoci di richiamare particolari funzioni volte alla decodifica della risorsa che si sta chiedendo.

Tra questi citiamo view-source: che permette di visualizzare il codice sorgente di una pagina web. Nell'esempio:

```
view-source:http://victim/index.html
```

3.5 Debugging del codice

Sviluppando un'applicazione web è indispensabile saper riconoscere, monitorare ed eventualmente risolvere problemi di progettazione. Ecco perché sono sempre più popolari, e per molti browser già integrati, i **Developers Tools**.

Si presentano come programmi integrati nel Browser e permettono di effettuare diverse operazioni:

- **Analizzare gli elementi HTML e gli stili CSS**, permettendo una modifica in live così da facilitarne lo sviluppo
- **Interagire con la Console**, una sorta di Terminale da cui è possibile stabilire funzioni in clientscript (Javascript)
- **Monitorare le Risorse**, ad esempio è possibile stabilire facilmente quali sono i file esterni caricati, l'impatto che hanno in termini di performance e di memoria

C'è da dire che i vari DevTools nativi o di terze parti offrono diverse soluzioni e strumenti diversi: nel nostro corso effettueremo dei test con Firefox che li integra nativamente, tuttavia sentiti libero di utilizzare il Web Browser che preferisci per aiutarti nello studio.

Di seguito elenchiamo alcuni Web Browser che supportano i DevTools o la relativa estensione di terze parti:

Browser	Dev Tools
Mozilla Firefox	Integrato
Google Chrome	Integrato
Opera Browser	Integrato
Safari	Integrato

È possibile richiamare la finestra dei DevTools effettuando un click destro sulla pagina, quindi cliccando su “**Ispeziona**” oppure “**Analizza Elemento**”.

I DevTools si presentano all’incirca in questo modo (figura X):

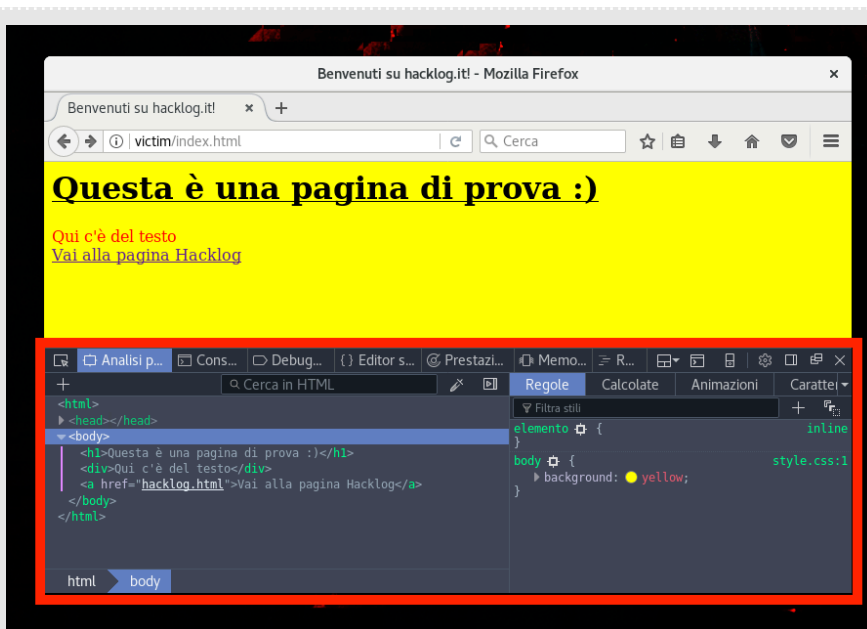



Figura X: esempio dell'analizza elemento di Firefox ESR su Debian 9

Per buona parte di questo testo potranno esserci utili i DevTools, quindi ti consiglio di darci uno sguardo (tranquillo, puoi smanettare quanto vuoi senza creare danni!) e riprendere la lettura quando sarai pronto.

3.6 Navigazione dinamica

Fino ad ora abbiamo visto come chiedere semplici pagine statiche al Web Server. Immagina ora di dover gestire un blog o un e-commerce: dovremmo creare ogni pagina HTML per ogni articolo? E se dovessimo modificare un elemento (tipo un logo) dovremmo modificare tutte le pagine?

Abbiamo bisogno di creare un'applicazione in grado di gestire queste e mille altre cose: a differenza dell'HTML o del Javascript dovrà poi essere in grado di interagire con il server, permettendo ad esempio di caricare immagini, salvare i dati – compresi i login e i dati di pagamento degli utenti – e altre operazioni che il browser non può – e non deve poter – fare.

L'ambiente in cui opererà l'applicazione è definito **Back-End**: se con HTML, CSS e Javascript abbiamo operato solo ed esclusivamente lato utente (client) questa volta sarà necessario lavorare lato servizi (server).

Allacciati le cinture dunque, adesso arriva il bello!

3.6.1 PHP

Il PHP è un linguaggio di programmazione concepito per la creazione di pagine web dinamiche: come abbiamo visto infatti l'HTML permette di generare pagine web statiche, che cioè non possono modificarsi in base agli input che gli da un utente.

Procediamo con ordine. Per prima cosa **installiamo PHP7** nella nostra Debian:

```
$ apt install php7.0 libapache2-mod-php7.0
```

Così facendo porteremo nella nostra macchina sia l'interprete di PHP, tutte le librerie e i moduli per poter lavorare con Apache2. Non dovrebbe essere necessario riavviare il Web Server, qualora ci fossero problemi ti invito a verificare lo status del Web Server con i comandi spiegati al capitolo 2.2 .

Come ormai siamo abituati a fare, riapriamo nano e creiamo un nuovo file per verificare il funzionamento del PHP:

```
$ nano test.php
```

e compiliamolo come segue:

```
<?php
    // Questo è un commento!
    echo("Hello, World!");
?>
```

Questo, signore e signori, è codice PHP! Analizziamolo come abbiamo fatto con gli altri:

- **Il codice PHP si apre e si chiude** con i tag <?php (spesso si potrà trovare anche solo <? e ?>

- **Il codice PHP si commenta** come il CSS, utilizzando // per una sola riga o /* e */ per più righe
- **Il codice PHP**, esattamente come il Javascript, **chiude ogni riga** con il ;
- **Le pagine PHP** hanno solitamente l'estensione **.php**

Caricando l'URL http://victim/test.php noteremo come, in realtà, vedremo a schermo stampato solo "Hello, World!"; a differenza di un linguaggio di mark-up, e come per il Javascript, abbiamo fatto uso di una funzione – in questo caso echo – che permette di stampare a video un messaggio da noi specificato, mentre tutto il resto viene elaborato da un interprete.

Come per il Javascript anche il PHP vuole le stringhe tra apici (') e virgolette ("). Non dimenticare MAI questo concetto!

Nota avanzata: Come vedi tra le due parentesi abbiamo usato i doppi apici. Questi simboli stanno a significare che al suo interno stiamo racchiudendo una stringa: senza di esse infatti il programma andrà in errore. Se tuttavia dovessi sentire l'esigenza di usare dei doppi apici puoi utilizzare il backslash (\) di fronte ad essa, ad esempio: echo(" Voglio usare le \"Virgolette\" ");

La probabilità di compiere errori durante la scrittura, specie se è la tua prima volta, è molto elevata! Basta una virgola in meno o uno spazio non visto e la pagina risulterà completamente bianca. Per questo ti consiglio di abilitare il debugger interno a PHP modificando il file php.ini:

```
$ nano /etc/php/7.0/apache2/php.ini
```

Con [CTRL+W] puoi ricercare un termine (dovrai trovare display_errors) e modificarne il valore da "Off" a "On". Se hai dubbi sulla riuscita dell'operazione prendi in considerazione la figura X. Una volta fatto procedi al riavvio del web server:

```
$ service apache2 restart
```

```
  GNU nano 2.7.4            File: /etc/php/7.0/apache2/php.ini            Modificato

; sending them to STDOUT.
; Possible Values:
;   Off = Do not display any errors
;   stderr = Display errors to STDERR (affects only CGI/CLI binaries!)
;   On or stdout = Display errors to STDOUT
; Default Value: On
; Development Value: On
; Production Value: Off
display_errors = On

; The display of errors which occur during PHP's startup sequence are handled
; separately from display_errors. PHP's default behavior is to suppress those
; errors from clients. Turning the display of startup errors on can be useful in
; debugging configuration problems. We strongly recommend you
; set this to 'off' for production servers.
; Default Value: Off
; Development Value: On
; Production Value: Off

^G Guida      ^O Salva      ^W Cerca      ^K Taglia     ^J Giustifica ^C Posizione
^X Esci       ^R Inserisci  ^\ Sostituisci^U Incolla    ^T Ortografia ^_ Vai a riga
```

Figura X: non impazzire cercando errori, fai in modo che sia il web server a dirti cosa non va!

Se dovessi avere problemi con la programmazione troverai al capitolo X il codice completo della piccola applicazione che a breve svilupperemo.

Nota avanzata: non volevo confonderti ulteriormente ma è doveroso anticipare quanto segue: PHP all'inizio era un linguaggio con paradigma procedurale (quello che useremo in questo testo) ma con la versione 5 ha integrato anche la programmazione ad oggetti (OOP). La programmazione ad oggetti è il modo migliore se si vuole progettare applicazioni complesse; inoltre molti programmatori fanno uso di framework, delle strutture logiche di supporto che facilitano di molto il lavoro di sviluppo, anticipando le necessità che il programmatore avrà.

3.6.2 PHP e HTML, un matrimonio che s'ha da fare

Da una parte abbiamo il PHP, dall'altro l'HTML; due linguaggi possono coesistere in maniera armoniosa poiché hanno scopi e princìpi diversi.

Il **PHP può generare codice HTML** – e non viceversa – quindi spesso ti troverai a lavorare con soli file di tipo .php . Riprendiamo il file test.php e proviamolo da noi:

```
<?php
    echo("
            <html>
                    <head>
                            <title>Ciao a tutti!</title>
                    </head>
                    <body>
                            Questo è HTML
                    </body>
            </html>
    ");
?>
```

Il risultato che ci si presenterà sarà il medesimo di una normale pagina HTML.

Stufo di nano, vero? Se hai installato Debian 9 con GNOME (come consigliato nella guida d'installazione) e preferisci l'ambiente grafico puoi usare gedit (su GNOME3) o pluma (su MATE), un editor di testo con GUI, quindi più nelle tue corde se non sei abituato con i Terminali; in questo caso puoi lanciare il comando gedit /var/www/html/test.php

---

Come abbiamo anticipato un linguaggio back-end (come PHP) può anche interagire con i file presenti nel Web Server. Dunque, potremmo decidere di includere un file esterno nel nostro mini-programma, aggiungendo alla fine dell'echo il seguente codice:

```
</html>
    ");
    include("hacklog.html");
?>
```

Se tutto è stato seguito alla lettera otterrai tra le mani una pagina web contenente codice esterno.

Questa logica sta alla base delle **pagine dinamiche**: se dovessi avere 100 pagine, tutte che includono un'unica pagina esterna (come un menù), e volessi cambiarne il contenuto, ti basterebbe cambiarla solo dalla pagina inclusa!

3.6.3 Una pagina di login? Ma certo!

HTML e PHP sono un'accoppiata micidiale quando c'è da scambiarsi informazioni tra di loro. Come si fa?

Iniziamo creando un nuovo file:

```
$ nano login.php
```

In questo esempio simuleremo una pagina di login, facendo in modo che sia in grado di auto-inviarsi delle informazioni e creando a tutti gli effetti un'applicazione "pensante"; la pagina sarà quindi http://victim/login.php

Compiliamo la pagina come segue:

```
<html>
    <body>
        <form name="login" method="get"
        action="login.php">
            <input type="text" name="password" />
            <input type="submit" value="Accedi" />
        </form>
    </body>
</html>
```

Spieghiamo in breve le varie tag presenti:

- **<form>**, questa tag viene utilizzata per comunicare all'HTML che "tutto quello che è qui dentro può essere inviato". Nota la presenza dell'attributo **action**, che permette di specificare a quale pagina inviare i dati (in questo caso la stessa in cui ci troviamo). Parleremo invece del **method** più tardi.
- **<input>**, ne abbiamo usati due: questi tag permettono di far interagire l'utente in maniera più marcata. Nel **type** abbiamo usato text (per indicare che l'utente

può inserire testo) e submit (per mostrare un pulsante che invia il contenuto del form)

Uhm... non è molto sicuro però lasciare la password in chiaro, non trovi? Facciamo un piccolo cambiamento alla input text, sostituendo il type in password:

```
<input type="password" name="password" />
```

Occhio a non fare confusione tra il type e il name! Con il primo si identifica il tipo di input (che sono standard HTML) mentre con name puoi scegliere tutto quello che preferisci :)

3.6.3.1 TRASFERIMENTO DEI DATI

La logica di questa pagina si traduce in: *invia tutto ciò che è dentro il form alla pagina login.php (che poi è se stessa).*

Ma come facciamo a recuperare i dati? Ma con il PHP ovviamente!

```
<html>
    <body>
        <?php
            if(isset($_GET['password']))
            {
                $password = $_GET['password'];
                echo("La tua pass è: " . $password);
        }
        ?>
        <form name="login" method="get"
        action="login.php">
            <input type="text" name="password" />
            <input type="submit" value="Accedi" />
        </form>
    </body>
</html>
```

Il poco codice che abbiamo scritto dirà:

1) Verifica **se** esiste nella query-string (a breve vedremo di che si tratta) se esiste il parametro "password"; se esiste, crea una **variabile** (chiamata $password) e salva in essa il valore che recuperi dal GET (tra poco ti dirò di cosa tratta). Nota che le variabili in PHP si creano mettendo il **simbolo dollaro** ($) di fronte alla nome della variabile che si vuole creare.

2) **Stampa a video** la stringa "La tua pass è: " seguita dalla variabile $password. Nota l'utilizzo del punto (.) che "concatena" due tipi di valori diversi (una stringa e una variabile). Questo è fondamentale, altrimenti otterresti un errore.

Prima di proseguire vorrei farti notare una cosa curiosa: nel codice che abbiamo scritto ci sono almeno un paio di vulnerabilità capaci di compromettere la sicurezza dell'applicazione! Trattieni i bollori, ne riparleremo a tempo debito :)

3.6.3.3 DICHIARAZIONI IF, ELSEIF E ELSE

Nell'universo della programmazione è fondamentale sapere se esistono le condizioni di poter fare qualcosa oppure no.

La struttura delle dichiarazioni si basa su questo semplice principio:

Logica	Programmazione
Se questa condizione è vera	IF
Se quest'altra condizione è vera	ELSEIF
Se nessuna delle condizioni precedenti viene soddisfatta	ELSE

Cerchiamo di fare qualche facile esempio per capire a cosa ci riferiamo.

Una dichiarazione base è data dalla **condizione IF** come la seguente:

```
SE <CONDIZIONE>
{
    FAI QUALCOSA
}
```

Quindi SE ci sono le condizioni che il programma si aspetta allora FAI QUALCOSA: questo "FAI QUALCOSA" è sempre indicato dentro le {parentesi graffe}.

Una dichiarazione molto semplice può essere:

```
if($numero == 1)
{
    echo("Il numero che hai scelto è 1");
}
```

Possiamo però voler dire anche di fare qualcos'altro SE una condizione non viene rispettata. Questo si fa con la **condizione ELSE**, quindi diremo:

```
SE <CONDIZIONE>
{
    FAI QUALCOSA
}
ALTRIMENTI
{
    FAI QUALCOS'ALTRO
}
```

Che in programmazione PHP risulterà:

```
if($numero == 1)
{
    echo("Il numero che hai scelto è 1");
}
else
{
    echo("Il numero che hai scelto non è 1");
}
```

In questo modo possiamo prendere nuove decisioni qualora la condizione specificata non è vera. E se volessimo verificare se un'altra condizione è vera? Useremo la **condizione ELSEIF**:

```
SE <CONDIZIONE>
{
```

```
    FAI QUALCOSA
}
OPPURE <CONDIZIONE>
{
    FAI QUALCOS'ALTRO
}
ALTRIMENTI
{
    FAI QUALCOS'ALTRO ANCORA
}
```

Che tradotto in linguaggio PHP sarà invece:

```
if($numero == 1)
{
    echo("Il numero che hai scelto è 1");
}
elseif($numero == 2)
{
    echo("Il numero che hai scelto è 2");
}
else
{
    echo("Il numero che hai scelto non è né 1 né 2");
}
```

È molto importante saper usare le condizioni: senza di esse non sapremmo come il programma ragiona e decide di compiere azioni piuttosto che altre.

Dati questi concetti, andiamo a migliorare il nostro codice come segue:

```
<html>
    <body>
            <?php
                    if(isset($_GET['password']))
                    {
                            $password = $_GET['password'];
```

```
                        echo("La tua pass è: " . $password);
            }
            else
            {
                    echo("Non hai specificato alcuna pass");
            }
            ?>
            <form name="login" method="get"
            action="login.php">
                    <input type="password" name="password" />
                    <input type="submit" value="Accedi" />
            </form>
        </body>
    </html>
```

Nota avanzata: se sei un programmatore PHP probabilmente storcerai il naso per la poca "cura" con cui abbiamo controllato l'input. Approfondiremo gli argomenti nei capitoli successivi.

3.6.3.3 METODI GET E POST

Come ricorderai nel cap. 2.4.1 URL abbiamo accennato alle query-string. Questa feature permette di passare in chiaro le informazioni da una pagina all'altra: basti infatti vedere il risultato una volta compilato l'input:

```
http://localhost/login.php?password=h4ckl0g
```

Questo, come avrai capito, non è proprio un bene quando c'è da passare informazioni di una certa riservatezza.

Il passaggio dei valori di un form attraverso la query-string avviene a causa del method="get" visto nel capitolo precedente (2.6.3.1 Trasferimento dei Dati); inoltre, se nel form non viene specificato alcun method, l'HTML di default utilizzerà il **GET**.

Per evitare che ciò accada possiamo sostituire il valore dell'attributo method con **POST**; in questo modo invieremo i dati senza che l'URL ne mostri il contenuto:

```
<html>
    <body>
        <?php
            if(isset($_POST['password']))
            {
                $password = $_POST['password'];
                echo("La tua pass è: " . $password);
        }
        else
        {
            echo("Non hai specificato alcuna pass");
        }
        ?>
        <form name="login" method="post"
        action="login.php">
            <input type="password" name="password" />
            <input type="submit" value="Accedi" />
        </form>
    </body>
</html>
```

3.6.3.2 COOKIE

Occorre prima di tutto specificare che il protocollo HTTP è stateless, ovvero che dopo ogni richiesta ad una pagina web, il server "dimentica" l'identità di un utente.

Una delle soluzioni che le applicazioni usano per memorizzare l'identità di un utente è attraverso i **cookies**, piccoli file che risiedono nel computer client e che possono essere letti dal web server.

Spieghiamolo con un po' di codice, aggiungendo dopo la dichiarazione della variabile:

```
...
    $password = $_POST['password'];
    setcookie("nome_cookie",$password);
    echo("La tua pass è: " . $_COOKIE['nome_cookie']);
...
```

Purtroppo come potrai notare compilando il form bisognerà ricaricare nuovamente la pagina prima di visualizzare la password in chiaro: questo succede poiché il linguaggio PHP considera lo stato iniziale del browser e non le modifiche che intercorrono.

Più semplicemente, quando carichiamo nuovamente la pagina il browser non ha ancora il cookie: questo viene creato durante lo script (con la funzione **setcookie**) mentre **$_COOKIE[]** si aspetta un cookie prima ancora dell'avvio applicazione!

Ci sono diverse soluzioni per ovviare al problema, tuttavia per i fini del nostro apprendimento non è necessario approfondire ulteriormente l'argomento (nulla che un buon libro sulla programmazione PHP o una guida in rete possa risolvere).

3.6.3.3 SESSIONI

Analogamente ai cookie, le **sessioni** vengono in sostegno al programmatore che vuole memorizzare informazioni di vario tipo, salvandole questa volta nel server anziché nel client.

A differenza del cookie però la sessione muore con la chiusura della tab del browser, mentre il cookie può rimanere salvato nel browser per un tempo indeterminato. Il funzionamento è tuttavia simile, sebbene cambino le funzioni:

```
...
    $password = $_POST['password'];
    session_start();
    $_SESSION['nome_sessione'] = $password;
    echo("La tua pass è: " . $_SESSION['nome_sessione']);
...
```

In questo caso si indica a PHP di collezionare le sessioni (**session_start**), di salvare la password dentro la sessione (**$_SESSION[]**) e infine di stamparla.

Come potrai notare il "difetto" di programmazione della sessione non si presenta come nel cookie: ciò è dovuto dal fatto che il server anticipa già le sessioni prima ancora di assegnarle, per questo motivo vedrai la password in chiaro sin da subito.

Sia setcookie che session_start sono funzioni che devono essere evocate prima di qualunque stampa a video di qualsiasi cosa, altrimenti il codice semplicemente non funzionerà! Tienilo a mente qualora volessi "giocherellare" un po' con le tue pagine web :)

3.6.3.4 LA NOSTRA PRIMA APPLICAZIONE WEB

Urrah! Abbiamo appena creato la nostra applicazione web! Se dovessi avere problemi con il codice (qualcosa che non funziona) ti lascio qui **l'intero codice scritto** fino ad ora:

```
<html>
    <body>
        <?php
            if(isset($_POST['password']))
            {
                $password = $_POST['password'];
                session_start();
                $_SESSION['nome_sessione'] =
$password;
                echo("La tua pass è: " .
                $_SESSION['nome_sessione']);
            }
            else
            {
                echo("Non hai specificato alcuna
pass");
```

```
                    }
            ?>
            <form name="login" method="post"
            action="login.php">
                    <input type="password" name="password" />
                    <input type="submit" value="Accedi" />
            </form>
    </body>
</html>
```

Con poche righe siamo riusciti a progettare una piccola applicazione web in grado di leggere un input utente (la password), stamparla a video e salvarne le credenziali sia su un browser che sul server.

Questo è, in soldoni, la programmazione in PHP. Niente male, non è vero? :)

3.7 Database

L'ultimo elemento che compone un'applicazione web è il Database: costruendo una qualunque applicazione arriverà presto o tardi la necessità di salvare dei dati per poi recuperarli successivamente e in qualsiasi momento.

Con un Database è possibile conservare **diverse tipologie di informazioni**, ad esempio i dati di login di un utente, il contenuto delle pagine di un sito e molto altro: per un cyber-criminale avere accesso a questo contenitore significherebbe ottenere il Santo Graal!

Il Database è quindi gestito da un **DBMS** che si occupa di *memorizzare, manipolare e collegare* dei dati: se in passato era sufficienti dei semplici file di testo col tempo la complessità e la crescita di questi contenitori hanno richiesto la necessità di sviluppare ambienti appositamente pensati per lo scopo.

Negli anni le diverse software house hanno progettato i loro DBMS per venire incontro alle esigenze del mercato degli sviluppatori: esistono DBMS proprietari o a pagamento (come Oracle Database e Microsoft SQL Server) a

quelli free e proprietari come MySQL, PostgreSQL, Percona, SQLite e via discorrendo.

Nel nostro caso faremo uso di **MariaDB**, un DBMS nato dalle ceneri di MySQL e ritenuto oggi come il successore spirituale di quest'ultimo.

Per installarlo, con tutte le dipendenze del caso, lanciamo:

```
$ apt install mariadb-client mariadb-server php7.0-mysql
```

Ci verrà chiesto se vogliamo di riconfigurare il Server Web: selezioniamo Apache con [SPAZIO], spostiamoci su OK [TAB] e confermiamo [INVIO].

Come per Apache possiamo verificare il funzionamento del servizio lanciando:

```
$ service mysql status
```

Ricevendo il solito messaggio:

```
mariadb.service - MariaDB database server
   Loaded: loaded (/lib/systemd/system/mariadb.service; enabled; vendor preset:
   Active: active (running) since Wed 2018-02-28 16:23:44 CET; 3min 9s ago
 Main PID: 7118 (mysqld)
   Status: "Taking your SQL requests now..."
 ...
```

Se presente alla terza riga lo status Active: active (running) [figura X] significa che il processo è correttamente avviato. Se così non fosse puoi riavviarlo lanciando il comando:

```
$ service mysql restart
```

sostituendo l'ultimo parametro con *start* e *stop* per le medesime azioni.

```
root@hacklog:/var/www/html# service mysql status
● mariadb.service - MariaDB database server
                                  md/system/mariadb.service; ena
  Active: active (running)        ince Thu 2018-06-07 18:43:02
                                  t=/bin/sh -c systemctl unset-
  Process: 11428 ExecStartPost=/etc/mysql/debian-start (co
  Process: 11312 ExecStartPre=/bin/sh -c [ ! -e /usr/bin/g
  Process: 11308 ExecStartPre=/bin/sh -c systemctl unset-e
  Process: 11305 ExecStartPre=/usr/bin/install -m 755 -o m
 Main PID: 11401 (mysqld)
   Status: "Taking your SQL requests now..."
    Tasks: 26 (limit: 4915)
   CGroup: /system.slice/mariadb.service
           └─11401 /usr/sbin/mysqld
```

Figura X: il servizio mariadb è attivo e funzionante

3.7.1 Tabelle, Righe e Colonne

La struttura su cui si basa un Database è fondamentalmente incentrato sui concetti delle tabelle, righe e colonne.

Devi sapere che ogni database al suo interno contiene una o più **Tabelle**: la loro natura è quella solitamente di "categorizzare" il contenuto delle informazioni. Se prendiamo ad esempio un Forum all'interno del database troveremo diverse tabelle: una per gli utenti, una per i post, una per le sezioni e via dicendo.

--- SCHEMA DI VARIE TABELLE ---

Come per le normali tabelle grafiche (esatto, quelle che fai con Excel, con un editor di testo oppure come ti hanno insegnato a scuola con carta e penna) esse sono fatte di *righe (record o row)* e di *colonne (column)*.

Prendiamo come esempio il seguente schema:

Tabella: utenti

id	username	password	email	anni
1	admin	h4ckl0g%21	admin@hacklog.it	28

🔑 id	username	password	email	anni
2	murder	NvLSfn)6da_	murdercode@hacklog.it	29
3	stefano	s9lli4"1ew*@	s.novelli@inforge.net	69

In questo esempio possiamo dire che:

- La **tabella** si chiama utenti
- Esistono 5 **colonne** che rappresentano i valori contenuti nelle righe
- Esistono 3 righe che contengono i valori

3.7.2 L'importanza dell'ID

Dovrebbe essere tutto chiaro, ad eccezione probabilmente dell'**id**: questa colonna è solitamente rappresentata da **numeri interi** (in gergo INT) che si **auto-incrementa** ogni volta che viene creato un nuovo valore.

Immagina che un utente crei un nuovo account nella tabella appena vista: l'applicazione web scriverà nella tabella una nuova riga, quindi aggiungerà un nuovo id (4) calcolato in base al precedente:

Tabella: utenti

🔑 id	username	password	email	anni
1	admin	h4ckl0g%21	admin@hacklog.it	28
2	murder	NvLSfn)6da_	murdercode@hacklog.it	29
3	stefano	s9lli4"1ew*@	s.novelli@inforge.net	69
4	novelli	nvlSeW@FJm	novelli.s@hacklog.it	96

Avrai inoltre notato che, affianco a id, è posta una chiave (🔑): nei Database questa icona rappresenta la **chiave primaria**, un campo speciale che identifica in maniera univoca una riga nel database.

Una chiave primaria ci permette di sapere con precisione di quale elemento stiamo parlando: se dovessimo avere degli omonimi nei rispettivi campi non sapremmo di quale specifico valore si tratta.

Prendiamo l'esempio di un'altra tabella:

Tabella: post

id	titolo	testo
1	Ciao a tutti!	Mi chiamo murder e ho 28 anni
2	Salute!	Mi chiamo Stefano!
3	Ciao a tutti!	Mi piace l'arrampicata :)
4	Salute!	Mi chiamo Stefano!

Come vedi i post con ID 2 e 4 hanno il contenuto simile: come fa l'applicazione a sapere quale dei due rimuovere, quando ad esempio dovessimo cliccare sul relativo pulsante? Non potrebbe!

3.7.3 Relazioni tra Tabelle

Consideriamo la seguente situazione:

--- SCREEN CON UTENTI E POST NON COLLEGATI ---

Negli esempi che abbiamo appena fatto non ci sono dei modi per capire quale utente ha scritto un post; inoltre, anche aggiungendo una colonna (es: autore) alla tabella post si presenterebbe un problema non da meno: se dovessimo modificare o rimuovere l'utente dalla tabella utenti il suo nome rimarrebbe in quello dei post!

--- SCREEN CON UTENTI E POST CON IL CAMPO AUTORE ---

Il modello più diffuso dei DBMS – compreso quello di MariaDB – viene detto di tipo **relazionale**: in buona sostanza ci è permesso di collegare le informazioni tra loro attraverso delle relazioni.

Vediamo come questo sia possibile:

--- SCREEN CON UTENTI E POST COLLEGATI DA CHIAVE PRIMARIA ---

Questo schema relazionale viene chiamato di tipo **uno-a-molti**: significa cioè che un solo utente (**uno**) può scrivere tanti post (**molti**) e non viceversa; infatti, lo stesso post (inteso con quella id primaria, non come contenuto ricorda!) non può essere scritto da più utenti.

Esistono anche relazioni **uno-a-uno** e **molti-a-molti**: perché non provi a pensare a quali situazioni si potrebbero creare con queste relazioni?

3.7.4 Il nostro primo database

In questo capitolo vedremo come creare un database, predisporre delle tabelle e strutturarle in modo da poterci essere utili per eseguire diverse operazioni all'interno della web application.

Operando da Terminale è possibile **connettersi al DBMS** attraverso il client di mysql precedentemente installato:

```
$ mysql -u root -p
```

Creare un database è un'operazione semplice. Basta lanciare il comando:

```
CREATE DATABASE forum;
```

dove social sarà il nome del database. Lanciando il comando (query) si potrà capire che è andato tutto secondo i piani in quanto il DBMS risponderà con:

```
Query OK, 1 row affected (0.00 sec)
```

In caso risponda ERROR... verifica che non ci siano errori di grammatica o dimenticanze (occhio soprattutto ad apici e punti e virgola).

A questo punto sarebbe opportuno **creare un utente** all'interno del DBMS capace di lavorare all'interno del Database. Usare l'account root per operare nel DB sarebbe un rischio troppo elevato e non consigliabile in nessuna circostanza:

```
CREATE USER 'user'@'localhost' IDENTIFIED BY 'passw0rd';
```

In questo esempio abbiamo creato l'utente "user", in grado di lavorare all'interno di localhost, con la password "passw0rd". È il momento di dire al DBMS che ha i permessi per lavorare all'interno del nostro Database:

```
GRANT ALL PRIVILEGES ON forum.* to 'user'@'localhost';
```

Possiamo a questo punto ricalcolare i permessi, quindi uscire dal client mysql:

```
FLUSH PRIVILEGES;
quit
```

3.7.5 phpMyAdmin, l'amico dei Database

Operare in un Database attraverso il Terminale può risultare parecchio frustrante, specie per chi non è abituato ad utilizzarlo notte e giorno: ad ovviare a questo problema ci viene incontro **phpMyAdmin**, una delle tante interfacce grafiche che ci consentono di lavorare all'interno di un DBMS senza impazzire tra le mille funzioni che devono essere ricordate.

Uno dei vantaggi di phpMyAdmin è quello di essere onnipresente in quasi tutti gli hosting[1] (a base Linux): ciò significa che, quando sarà il momento di parlare di hosting esterni, ritroveremo molto probabilmente phpMyAdmin già preinstallato.

Al momento però non è presente nel nostro Sistema Operativo, quindi **installiamolo** con il comando:

```
$ apt install phpmyadmin
```

Ci verrà chiesto di inserire una password (o di farne generare una) con cui verrà creato l'account phpmyadmin da associare al DBMS; in seconda battuta ci verrà chiesto se vogliamo configurare il database di phpmyadmin con dbconfig-common: anche in questo caso scegliamo <Si> [INVIO].

L'interfaccia grafica sarà ora disponibile all'indirizzo http://victim/phpmyadmin

e si presenterà in questo modo come in figura X.

[1] Spazi web predisposti alla distribuzione di applicazioni web.

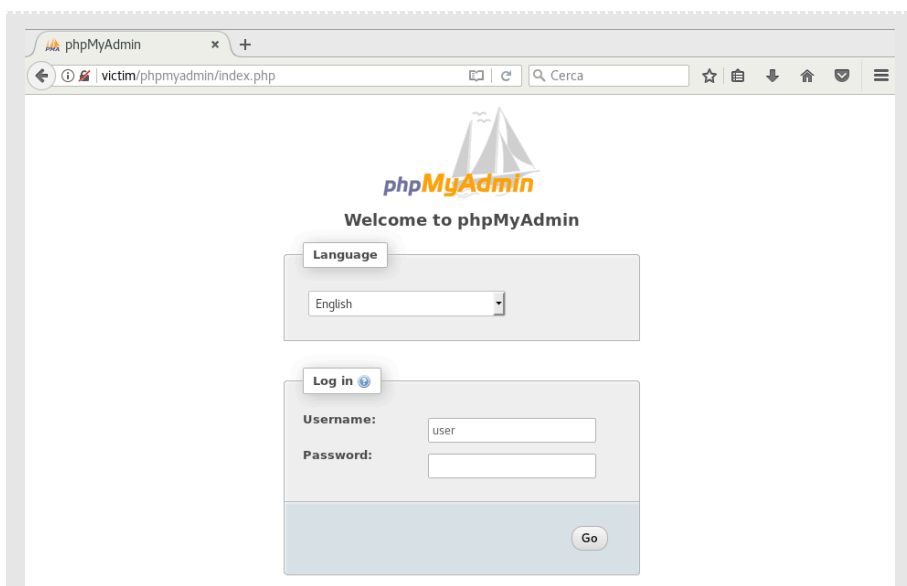



Figura X: schermata di login di phpMyAdmin

Procediamo al login con l'account creato nel capitolo precedente (nel nostro caso user e passw0rd) e verremo rifiondati nella dashboard iniziale come in figura X.

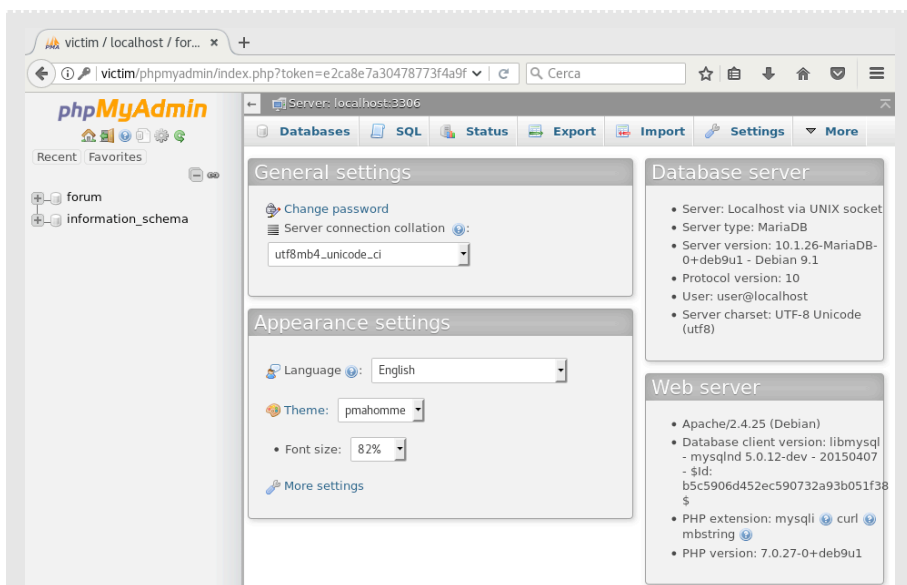



Figura X: dashboard iniziale di phpMyAdmin

3.7.5.1 CREAZIONE DI UNA TABELLA

Lo scopo di questo capitolo è quello di creare un piccolo Database in grado di funzionare con un'applicazione web progettata da noi, per poi capire in che modo questa può essere violata.

Da phpMyAdmin selezioniamo il database forum (nella colonna sinistra), quindi nella tab Structure (tasto presente in alto) ci verrà comunicato che non esistono tabelle, proponendoci di crearne una. Compiliamo il form indicando il nome (utenti) e il numero di colonne (5) come in Figura X.

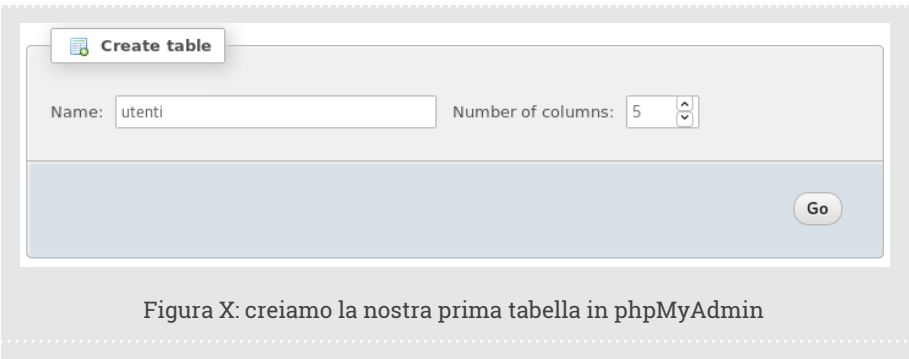
Figura X: creiamo la nostra prima tabella in phpMyAdmin

Andremo ora a ricreare la struttura della tabella vista in precedenza, che ti riporto per motivi di comodità:

Tabella: utenti

id	username	password	email	anni
1	admin	h4ckl0g%21	admin@hacklog.it	28

Dal form di creazione di una tabella andremo a compilare come segue (se hai dubbi fai riferimento alla Figura X), quindi ricordati di salvare le modifiche col tasto Save (in fondo a destra). Spiegheremo più avanti i significati di Type e Lenght/Values, per ora attieniti a seguire alla lettera lo schema:

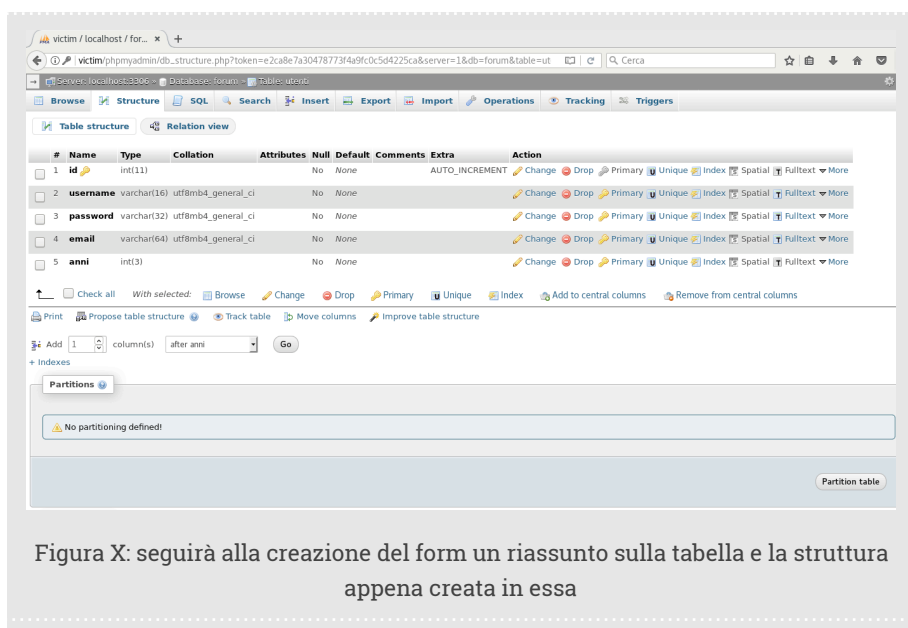
Name	Type	Lenght/Values	Index	A_I
id	INT	11	PRIMARY	X
username	VARCHAR	16		
password	VARCHAR	32		

Name	Type	Lenght/Values	Index	A_I
email	VARCHAR	64		
anni	INT	3		

Ricorda che il primo campo (id) dovrà essere una chiave primaria auto-incrementale (in phpMyAdmin il campo è A_I): in ogni caso spuntando la checkbox dell'A_I il client si preoccuperà di consigliarci automaticamente una Index di tipo Primary.

Figura X: ecco come dovrebbe essere compilato il form alla creazione della tabella

Dopo aver compilato il form (figura X) clicchiamo su Save in fondo e verremo catapultati nel riassunto della tabella appena creata (figura X).

Figura X: seguirà alla creazione del form un riassunto sulla tabella e la struttura appena creata in essa

3.7.5.2 MANIPOLARE I VALORI

Nel precedente capitolo abbiamo visto come creare una tabella e, conseguentemente, come predisporre la "struttura" di un database indicando quali colonne creare. In questa parte del testo vedremo invece come inserire, modificare ed eliminare valori da un database.

Assicurati innanzitutto di essere all'interno della tabella utenti. Per farlo ti basta guardare nella barra grigia in alto, dovrà essere in questo modo:

```
Server: localhost:3306 » Database: forum » Table: utenti
```

Se non dovesse essere così clicca su "utenti" posto all'interno del database "forum" nel menù alla sinistra della pagina.

Nel nostro caso ci interessa creare una riga con dei valori specifici, quindi clicchiamo sulla tab "Insert" in alto e compiliamo i Value come segue (se hai dubbi fai riferimento alla Figura X):

id	(VUOTO)
username	admin
password	h4ckl0g%21

id	(VUOTO)
email	admin@hacklog.it
anni	28

Figura X: form di inserimento di un valore all'interno della tabella utenti

phpMyAdmin ci comunicherà l'avvenuta esecuzione; possiamo comprovarne l'esito positivo cliccando sulla tab Browse in alto, dove verremo riportati alla lista delle righe presenti nella tabella utenti (figura X).

	id	username	password	email	anni
Edit Copy Delete	1	admin	h4ckl0g%21	admin@hacklog.it	28
Edit Copy Delete	2	murdercode	39Adow02	info@murdercode.it	20
Edit Copy Delete	3	stefano9lli	asdrubale123	posta@stefano9lli.com	19
Edit Copy Delete	4	niuji	lfpsaRlq402!3	s.novelli@piratech.it	25

Figura X: la lista degli utenti appena creati nel nostro Database

Notiamo come il campo id assumerà, senza che noi dicessimo nulla, il valore 1: sentiamoci liberi di creare nuovi valori (senza mai specificare l'id), se la struttura è

corretta gli id successivi incrementeranno ogni volta di un'unità (2,3,4 e via discorrendo). Al fianco di ogni riga (nel nostro caso una sola) ci saranno tre icone, rispettivamente: **Edit** – **Copy** – **Delete**; ognuna di esse ci permetterà di eseguire la relativa azione.

3.7.6 Il linguaggio SQL

Come abbiamo visto interagire con un database SQL è possibile sia con una GUI grafica sia con il **linguaggio SQL**. Questo linguaggio è fondamentale per chiunque abbia pensato almeno una volta nella vita: "voglio bucare un sito web".

L'SQL (acronimo di Structured Query Language) è un linguaggio con cui è possibile comandare i database di tipo relazionale: possiamo ad esempio aggiungere nuovi valori o eliminarli, cercare all'interno di una riga, vedere quante ce ne sono in una tabella e così via, esattamente come possiamo fare da phpMyAdmin.

Il vantaggio di saper comandare in SQL è la facoltà di lavorare su un numero infinito di DBMS e quindi di non dipendere da alcun client; inoltre, come vedremo più avanti, sarà fondamentale saperlo utilizzare per estrapolare dati a seguito di una violazione di un portale web.

Fortunatamente l'SQL è un **linguaggio davvero semplice** da utilizzare: è infatti rappresentato attraverso termini inglesi di uso comune e ogni azione (in gergo "query") corrisponde ad una frase di senso compiuto.

Prendiamo ad esempio la query SELECT, una delle più popolari in questo linguaggio:

```
SELECT username FROM utenti WHERE username='admin';
```

che tradotto è praticamente:

```
SELEZIONA lo username DALLA TABALLE utenti IL CUI
username È UGUALE A 'admin';
```

Il database ci risponderà con uno o più risultati se:

1) Esiste la tabella utenti

2) Esiste la colonna username

3) Esiste almeno una riga il cui username è uguale a admin

3.7.6.1 SOPRAVVIVERE IN SQL

Come già accennato questa non vuole e non può essere una guida completa all'SQL, quanto piuttosto un corso intensivo su quelle 4 o 5 cose da sapere per metter mano nei tuoi progetti.

Ecco allora che andiamo solo alcune delle funzioni SQL presenti, cercando di abituarti a quelle logiche che maggiormente ti troverai ad affrontare nel percorso della web security.

Rivediamo come si compone una query di tipo SELECT:

```
SELECT <colonna> FROM <tabella> WHERE <condizione>
```

Proviamo a lanciare una query sul nostro phpMyAdmin: selezioniamo il database (forum) dal menù a sinistra, quindi clicchiamo sulla tab SQL in alto e inseriamo la seguente query, quindi confermiamo con il tasto Go in basso a destra:

```
SELECT username FROM utenti WHERE username='admin';
```

Una piccola tabellina comparirà al centro della pagina (figura X).

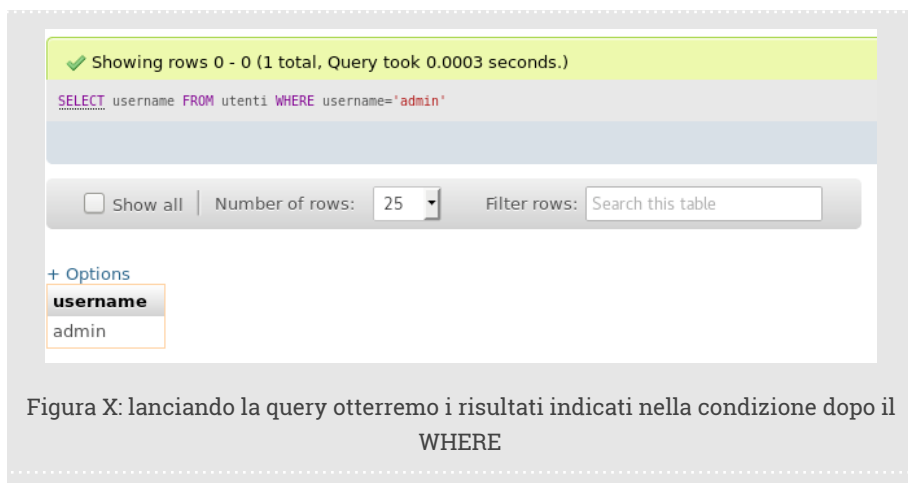



Figura X: lanciando la query otterremo i risultati indicati nella condizione dopo il WHERE

Nel caso volessimo visualizzare più colonne ci basterà specificarle, separandole con una virgola:

```
SELECT username, password, email FROM utenti WHERE
username='admin';
```

In alcune tabelle potrebbero esserci centinaia di colonne e sarebbe impossibile ricordarsele tutte! Ecco allora che possiamo utilizzare il carattere jolly asterisco (*); in questo modo potremo visualizzare tutti i campi:

```
SELECT * FROM utenti WHERE username='admin';
```

In una query di tipo SELECT è possibile applicare una condizione ad una colonna solo se questa è presente dopo la funzione SELECT. Se ad esempio lanciamo la seguente query il DBMS ci darà errore:
SELECT username FROM utenti WHERE password='passw0rd';
In questi casi saremo costretti a richiamare il campo password come segue:
SELECT username, password FROM utenti WHERE password='passw0rd';

Oltre alla SELECT esistono altre funzioni. In particolare ne emergono tre che permettono di lavorare nelle tabelle:

- **UPDATE**, che consente di modificare una riga
- **INSERT**, che consente di aggiungere una riga
- **DELETE**, che consente di eliminare una riga

Nello specifico esiste una struttura da seguire per ogni query. Vediamole assieme:

QUERY "INSERT"

Azione	SQL Query
Inserire una riga contenente i valori: username = "stefano" password = "123456" email = "s.novelli@inforge.net" anni = 26	INSERT INTO utenti (username, password, email, anni) VALUES ("stefano", "123456", "s.novelli@inforge.net", 26);

In questo caso consideriamo che venga creata una riga con chiave primaria (id) 5.

QUERY "UPDATE"

Azione	SQL Query
Modificare una riga, ad esempio la password e gli anni dell'utente appena creato	UPDATE utenti SET password='QWErty04', anni=25 WHERE id=5;

In questa circostanza notiamo come venga specificato l'id anziché lo username; in questo modo siamo certi di modificare quella riga.

QUERY "DELETE"

Azione	SQL Query
Eliminare una riga, ad esempio quella recentemente creata	DELETE FROM utenti WHERE id=5;

Così facendo elimineremo per sempre la riga appena creata, senza possibilità di recuperarla. Ricordati sempre la condizione WHERE: se non la specifichi eliminerai tutte le righe della tabella!

Ovviamente esistono molte altre funzioni tra cui: DROP (per eliminare intere tabelle!), REPLACE, RENAME, ALTER, SHOW, CREATE, OPTIMIZE e via dicendo. Queste e altre funzioni sono ben documentate sul sito https://www.w3schools.com/sql/

Come puoi vedere ogni query ha la sua struttura: purtroppo non esiste uno "schema" standard, quindi dovrai ricordartele a memoria (o googlarle!).

3.7.6.2 CONDIZIONI IN SQL

Cosa succede quando vogliamo verificare se una query risponde a più condizioni? Fino ad ora abbiamo visto infatti che:

```
SELECT <colonna> FROM <tabella> WHERE <condizione>
```

In programmazione tuttavia può essere utile verificare non una ma più condizioni, ad esempio:

```
SELECT <colonna> FROM <tabella>
```

```
WHERE <condizione> AND <condizione>
```

L'operatore AND, in questo caso, riporterà un risultato solo se entrambe le condizioni sono soddisfatte, ad esempio:

```
SELECT * FROM utenti
WHERE username='stefano' AND password='123456'
```

Allo stesso modo possiamo verificare se solo una delle due condizioni è vera:

```
SELECT <colonna> FROM <tabella>
WHERE <condizione> OR <condizione>
```

L'operatore OR ci permette quindi di ottenere un risultato se una delle due condizioni è vera:

```
SELECT * FROM utenti
WHERE username='stefano' OR password='h4ckl0g%21'
```

3.7.6.3 TIPI DI VALORI IN SQL

Se sei stato attento avrai notato come, in certe occasioni, i valori dopo l'uguale (=) sono sempre specificati tra virgolette, mentre in altri no. Rivediamo un istante la query di UPDATE precedente:

```
UPDATE utenti SET password='QWErty04',anni=25 WHERE id=5;
```

In questo caso la password deve essere tra 'apici' o "virgolette" (in questo caso non fa differenza), mentre questo non è necessario per anni e id. Il motivo?

Se ben ricorderai quando abbiamo creato la tabella, e di conseguenza creato la struttura che la compone, ad alcune colonne abbiamo specificato il valore VARCHAR, ad altre il valore INT:

Name	Type	Lenght/Values	Index	A_I
id	INT	11	PRIMARY	X
username	VARCHAR	16		
password	VARCHAR	32		
email	VARCHAR	64		
anni	INT	2		

Come in PHP e in Javascript, in SQL è necessario mettere le stringhe tra apici per evitare di confondere con i numeri e tutti gli altri elementi che si usano in programmazione. Ricordati bene questo concetto, lo troverai molto spesso nelle prossime pagine.

VARCHAR e INT sono due tipologie, tra le tante disponibili, che in SQL ci permettono di **determinare i tipi di valore** che in quella colonna sono consentiti: con VARCHAR possiamo far contenere qualunque carattere (VARCHAR, "set di caratteri variabile") mentre con INT possiamo far contenere solo numeri interi (INT, "interi").

Ogni tipo di valore a sua volta richiede (quasi) sempre una **lunghezza massima consentita**: nel nostro caso abbiamo detto a id di non superare gli 11 caratteri massimi. Questo non significa che id dovrà essere minore di 11, ma che non potrà avere più di 11 caratteri (quindi, raggiungerà un massimo di 99.999.999.999 valori); allo stesso modo la username non potrà avere più di 16 caratteri, la password 32 e così via...

3.7.7 PHP e i Database, la combo perfetta

PHP può tranquillamente collegarsi a MariaDB; esistono due diversi approcci – **MySQLi e PDO** – adatti al modo in cui è possibile programmare il PHP – procedurale o a oggetti. Nel nostro caso faremo uso del metodo MySQLi, in quanto più "nelle corde" con quanto visto.

Riprendiamo la nostra mini-applicazione scritta in PHP. Se non ricordi l'editor si apre con il comando:

```
$ nano login.php
```

e andiamola a modificare con il codice che segue (modificheremo solo la parte PHP, quindi che inizia con <?php e finisce con ?>).

A questo punto le cose potrebbero sembrare più complicate per la **mole di codice** che sto per mostrarti: prendi fiato, rileggi attentamente ciò che è scritto, analizza riga per riga ciò che ti spiegherò, cerca in rete se qualcosa non ti è chiaro e ritorna poi con calma.

Commenterò ogni riga di codice: troverai, come già spiegato, i commenti in questi due formati:

```
<?php
// Commento su una sola riga
/* Commento su più righe */
?>
```

Questa è una parte nevralgica per comprendere le vulnerabilità in un'applicazione web: non mollare proprio ora, dopo tutta la strada che hai fatto per arrivare qui! Ci vorrà un po' di pazienza, leggi con calma i commenti e cerca di comprendere del perché funzioni in questo modo. Sei pronto? Forza, cominciamo!

```
<html>
    <body>
<?php
    //Avvio il collezionamento sessioni
    session_start();
    //Verifico se in query-string è presente "password"
    if(isset($_POST['password']))
    {
        //Creo la variabile $password
        $password = $_POST['password'];
        //Mi connetto al database forum
        $conn = @mysqli_connect('localhost', 'user',
    'passw0rd', 'forum');
        //Se la connessione non va a buon fine
        if(!$conn)
        {
            //Chiudi tutto
            die("Impossibile connettersi al Database");
        }
        //Creo la variabile $sql con la query
        $sql = "SELECT username, password FROM utenti WHERE
password='$password'";
        //Eseguo la query nel database
```

```
            $query = mysqli_query($conn, $sql);
            //Raccolgo tutti i valori della query
            while ($row = mysqli_fetch_array($query))
            {
                   /* Creo la variabile $username che conterrà
il          valore "username" trovato nel database */
                   $username = $row['username'];
                   // Creo la sessione
                   $_SESSION['nome_sessione'] = $username;
            }
            // Chiudo la connessione al Database
            mysqli_close($conn);
     // Stampa $password, quindi torna a capo (<br />)
     echo("Hai digitato:" . $password . "<br />");
     }
     // Se la sessione esiste
            if(isset($_SESSION['nome_sessione']))
            {
                   // Stampo lo username a video
                   echo("Il tuo username è: " .
                   $_SESSION['nome_sessione']);
            } else {
                   echo("Non hai specificato alcuna pass
valida");
            }
     ?>
            <form name="login" method="post" action="login.php">
                   <input type="password" name="password" />
                   <input type="submit" value="Accedi" />
            </form>
     </body>
</html>
```

Compiliamo la input con una delle password salvate nel database e l'applicazione web risponderà con la nostra username! (Figura X)

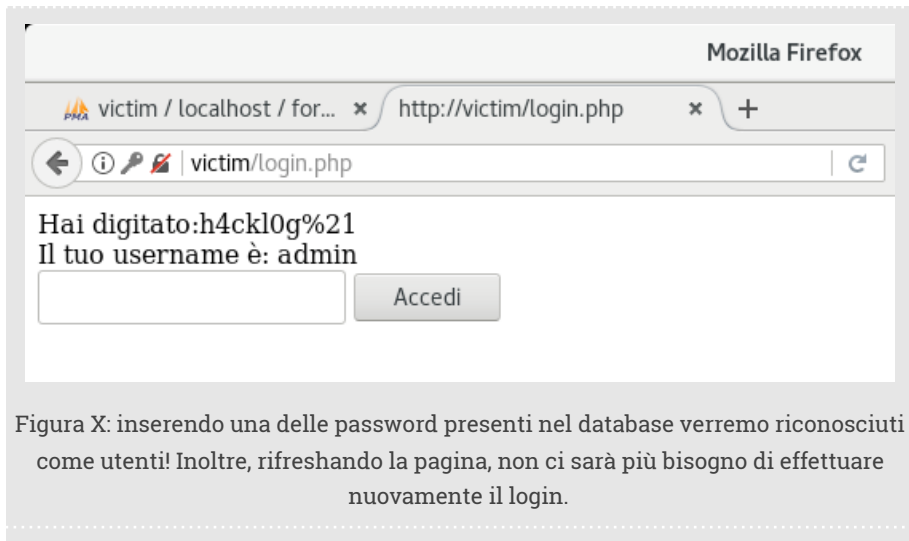



Figura X: inserendo una delle password presenti nel database verremo riconosciuti come utenti! Inoltre, rifreshando la pagina, non ci sarà più bisogno di effettuare nuovamente il login.

3.8 Il tuo primo Hack

È stata dura ma ce l'hai fatta, hai creato la tua prima applicazione web!

Lo studio sulla programmazione è finito (puoi stapparti una bella birra se vuoi festeggiare!) tuttavia quella che hai avuto modo di testare è solo una piccolissima porzione del meraviglioso mondo che circonda la programmazione web.

Tutto questo a cosa ha portato, ti chiederai? Non dovrei ancora spoilerare nulla (c'è ancora tanta strada da fare) ma... ok, solo una (anzi due!) :)

All'interno del nostro codice sono presenti diverse vulnerabilità: noi lo sappiamo perché... beh, lo abbiamo sviluppato noi!

Di quelli presenti ne vedremo due in particolare (magari chiuso questo documento ti verrà voglia di ritornare qui e provare a cercarne altre), semplicissimo da eseguire e che richiede le conoscenze che hai appena appreso.

Codice Javascript arbitrario

In questo esempio vedremo come sfruttare una falla all'interno del codice PHP per eseguire codice Javascript arbitrario: sebbene non dimostri efficacemente la pericolosità della vulnerabilità, è abbastanza per capire dove nasce il pericolo (vedremo poi i rischi).

Ricordi la **input password**? Mi riferisco a questa:

```
<input type="password" name="password" />
```

Come ricorderai il suo valore viene passato in POST, quindi salvato in una variabile:

```
$password = $_POST['password'];
```

per poi essere stampato:

```
echo("Hai digitato:" . $password . "<br />");
```

Ma se, invece di una password nella input... mettessimo codice Javascript?

```
<script>alert('This website has been hacked!');</script>
```

Il web server mostrerà ciò che abbiamo digitato sul nostro browser! Questo succede poiché ciò che inseriamo nella input viene salvato nella variabile $password, quindi viene stampato a schermo.

L'applicazione web interpreterà allora il tutto come segue:

```
$password = "<script>alert('Hello, hax0r!')</script>";
echo("Hai digitato:" . $password . "<br />");
```

Facendo risultare il codice Javascript all'interno del codice sorgente della pagina: puoi provare usando tu stesso l'Analizza Elemento cercando la porzione di codice infetta (Figura X).

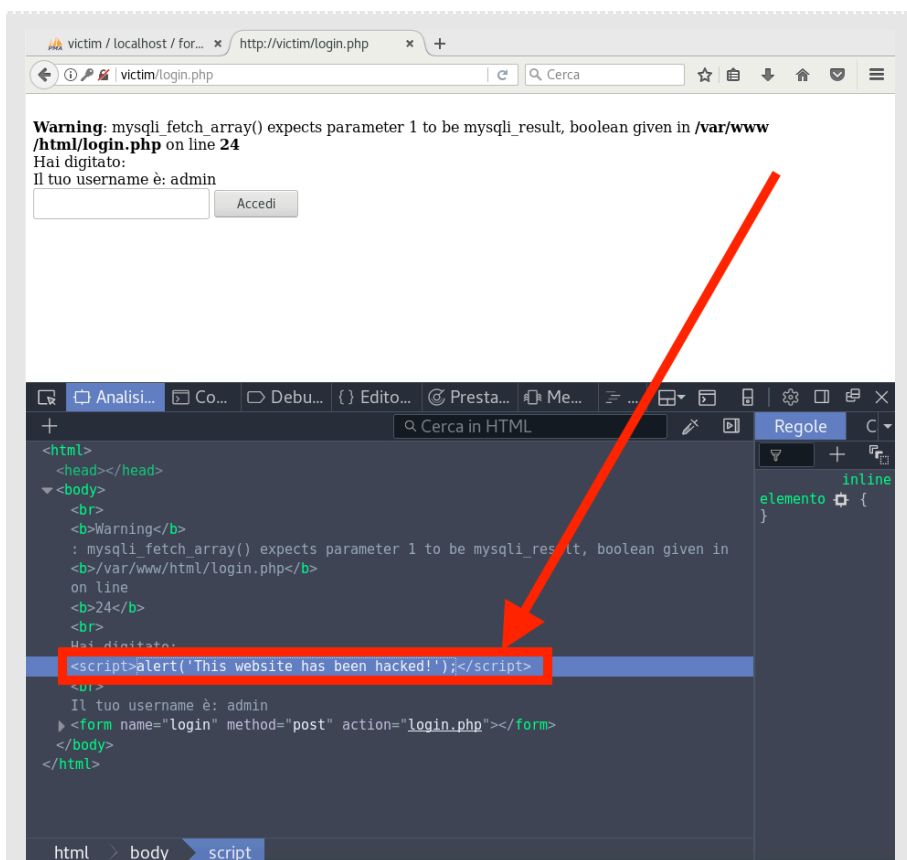



Figura X: inserendo codice Javascript nella Input causeremo l'esecuzione dello stesso.

Questa vulnerabilità, in Sicurezza Informatica, viene chiamata XSS (Cross Site Scripting) e permette di causare danni anche gravi in un portale web. Ne parleremo con molta calma al capitolo X.

Codice SQL arbitrario

Sulla scia del precedente esempio riprendiamo la porzione di codice:

```
$password = $_POST['password'];
```

Come ricorderai la password non solo viene stampata ma anche utilizzata per compiere una query:

```
$sql = "SELECT username, password FROM utenti WHERE
password='$password'";
```

Se scrivessimo pippo nella input ovviamente la query risulterebbe:

```
SELECT username, password FROM utenti WHERE
password='pippo';
```

Se scrivessimo ASDrubale! risulterebbe:

```
SELECT username, password FROM utenti WHERE
password='ASDrubale!';
```

Ora, ricordi cosa ti dissi dell'apostrofo? Serve ad aprire e chiudere una stringa, giusto? Se noi utilizzassimo un'apostro al posto della stringa, quello che ne seguirebbe diventerebbe codice SQL a tutti gli effetti, giusto?

Mi spiego meglio: se io scrivessi nell'input qualcosa tipo ' CODICESQL ' il risultato che ne verrebbe fuori sarebbe:

```
SELECT username, password FROM utenti
WHERE password='' CODICESQL'';
```

Quindi se noi inserissimo nella input del codice SQL preceduto dall'apostrofo (') come il seguente:

```
' OR '1'='1
```

Il risultato della query sarebbe [figura X]:

```
SELECT username, password FROM utenti
WHERE password='' OR '1'='1';
```

La query fa uso dell'operatore OR che, come abbiamo visto, permette di risolvere una query se una delle due condizioni è vera: poiché '1' è sempre uguale ad '1', la query riporterà un risultato vero.

Figura X: siamo stati riconosciuti come utenti del database senza conoscerne la password! Meraviglioso, non è vero?

Questa pericolosissima vulnerabilità permette di loggarsi come qualunque utente, eseguire codice SQL direttamente sul browser o eliminare il database con un solo comando: il suo nome è SQL Injection e ne parleremo al capitolo X.

3.9 CMS

Con l'avanzare di siti web si è reso necessario lo sviluppo di software che consentissero al web master di occuparsi solo della parte dei contenuti, lasciando così all'applicazione web l'arduo compito di far funzionare le cose.

Le necessità di avere un sistema di login con eventuali pannelli di controllo, la gestione di articoli o pagine, la creazione di menù e tutto ciò a cui siamo abituati per l'amministrazione di un portale è oggi affidato a un CMS.

Un CMS (più specificatamente Web Cms, Web Content Management System) è quindi composto dall'insieme di tecnologie viste precedentemente. Ad esempio **Wordpress**, il più popolare tra i CMS in circolazione (che sarà anche oggetto di una nostra analisi nella parte Sicurezza) è composto principalmente da:

- Codice *Back-End*, in grado di coprire la maggior parte delle esigenze di un utente, ivi comprese la gestione dei plugin e dei template; inoltre il codice PHP

effettua la prima connessione al Database e ne crea tutta la struttura da cui poi attingerà alle informazioni e ne consentirà la modifica, l'inserimento e l'eliminazione

- Codice *Front-End*, ricoprendo in toto tutta la parte dell'esperienza utente lato amministrazione e fornendo un template (base) per l'esperienza lato client attraverso codice HTML, CSS e Javascript

L'installazione di un CMS è spesso semplice e viene fornita di relativa documentazione, tuttavia la procedura più comune è la seguente:

1) Si caricano i file forniti dal sito del produttore nella cartella (o sottocartella) del web server; in una situazione con Hosting, si procederà al caricamento dei file nello spazio riservato tramite il programma adeguato (FileZilla, Cyberduck etc..) con il protocollo più adeguato (SFTP/FTPS o il più antico FTP)
2) Si crea manualmente il database che conterrà le informazioni del sito[1]
3) Si avvia la procedura di configurazione; questa viene affidata sempre al PHP che scriverà in un file le coordinate per il collegamento al database
4) Il CMS provvederà quindi a verificare che tutte le dipendenze siano soddisfatte, quindi scriverà nel database la struttura e le righe necessarie per il primo funzionamento

---

Attraverso i pannelli di gestione web offerti dagli hosting (es: CPanel, Plesk, Webmin, DirectAdmin etc...) è possibile comandare l'installazione di un CMS tramite una più semplice procedura di installazione one-click. Tali funzioni non sono sempre presenti (in quanto estensioni); le più popolari sono Installatron, Fantastico, Softaculous, SimpleScripts etc...

---

[1] Più recentemente sono aumentati i CMS di tipo NoSQL; grazie all'introduzione di memorie sempre più veloci e alle poche necessità che richiedono la gestione di questi portali (vedesi relazioni) si sta preferendo l'uso di CMS che non dipendono da DBMS.

I CMS hanno acquisito un'enorme popolarità, tanto da ricoprire esigenze di settore e disponibili in varie forme di distribuzione (opensource, free, hosted o a pagamento) com ad esempio:

- **Generali**: i CMS tutto fare, che offrono una base per poi essere espandibili grazie all'uso di addon di terze parti. L'esempio più popolare è ovviamente *Wordpress* (sebbene sia partito come un CMS per fare blog) ma c'è un ottimo riscontro del pubblico anche per *Joomla*, *Drupal*, *Jekyll*, *Ghost*, *Grav, TYPO3* e molti altri
- **Ecommerce**: i CMS progettati per l'acquisto online, in questa categoria troviamo *Prestashop, Magento, osCommerce* etc...
- **Forum**: chiamati anche FMS (Forum Management System) sono specializzati per la creazione di board di discussione. I più popolari sono: *vBulletin, Xenforo, phpBB, myBB, Burning Board, Flarum, Simple Machines Forum, nodeBB, IPBoard* etc...

Come scopriremo più avanti l'uso di un CMS facilita di molto il compito di metter su un portale web, tuttavia può lasciare il portale web esposto ad attacchi di diverse entità.

3.9.1 Damn Vulnerable Web Application (DVWA)

Nel corso di questo volume studieremo alcune vulnerabilità attraverso DVWA, un CMS specificatamente progettato per effettuare test di vulnerabilità ai danni di un'applicazione web. Questa web app è la stessa presente nel nostro WHLAB[1] e puoi tranquillamente usarla in questa situazione se dovessi avere problemi a configurare DVWA.

3.9.1.1 SCARICARE DVWA

[1] Maggiori info su www.hacklog.net

Concludiamo la preparazione dell'ambiente d'attacco scaricando DVWA dal sito ufficiale all'interno della cartella del web server nella macchina **vittima**:

```
$ cd /var/www/html
$ wget https://github.com/ethicalhack3r/DVWA/archive/master.zip
$ unzip master.zip
$ mv DVWA-master vuln
$ rm master.zip
```

Collegandosi all'indirizzo http://victim/vuln dalla macchina **attacker** riceveremo l'errore:

```
DVWA System error - config file not found. Copy config/
config.inc.php.dist to config/config.inc.php and
configure to your environment.
```

È arrivato il momento di configurare il CMS e renderlo funzionante a tutti gli effetti.

3.9.1.2 CONFIGURARE DVWA

La configurazione prevede nel collegare i file appena scaricati a un database che dobbiamo creare; le coordinate di una configurazione (come in quasi tutti i CMS) si esegue da un file chiamato config:

```
$ cd vuln
$ cp config/config.inc.php.dist config/config.inc.php
```

Prima di configurare il file creiamo un nuovo database in cui salveremo le informazioni di DVWA: i comandi sono gli stessi del capitolo X ma per comodità li rivedremo:

```
$ mysql -u root -p
> CREATE DATABASE dvwa;
> GRANT ALL PRIVILEGES ON dvwa.* to 'user'@'localhost';
> FLUSH PRIVILEGES;
> quit;
```

Raccomandiamo inoltre di configurare un reCAPTCHA (http://google.com/recaptcha) assegnandogli il dominio "victim" e copiando le chiavi (le useremo poi) [figura X]:

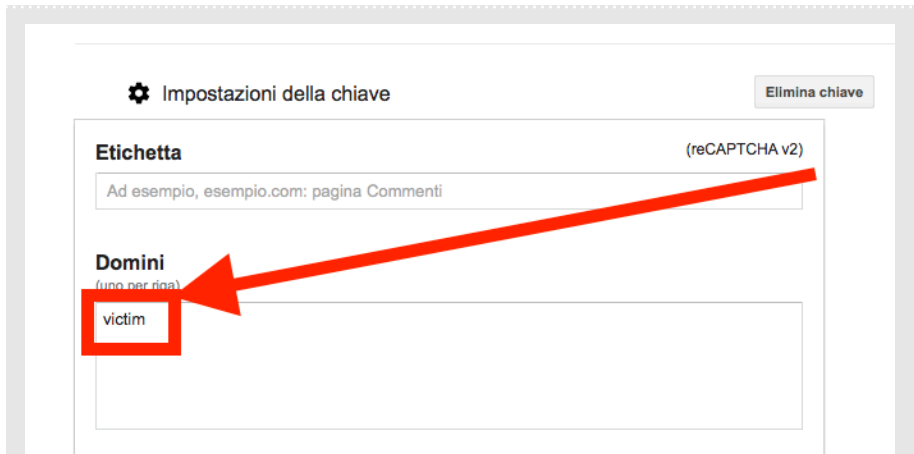



Figura X: aggiungi il dominio "victim" altrimenti reCAPTCHA si rifiuterà di funzionare

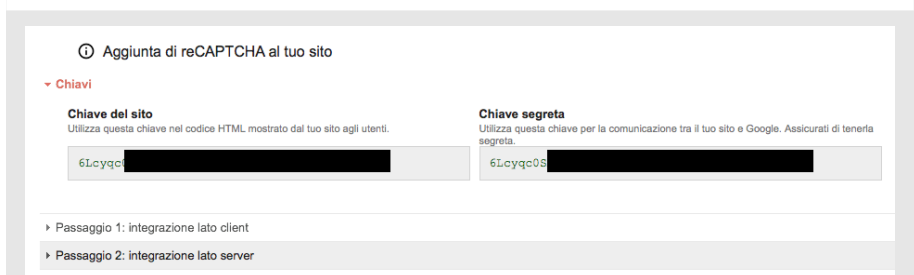
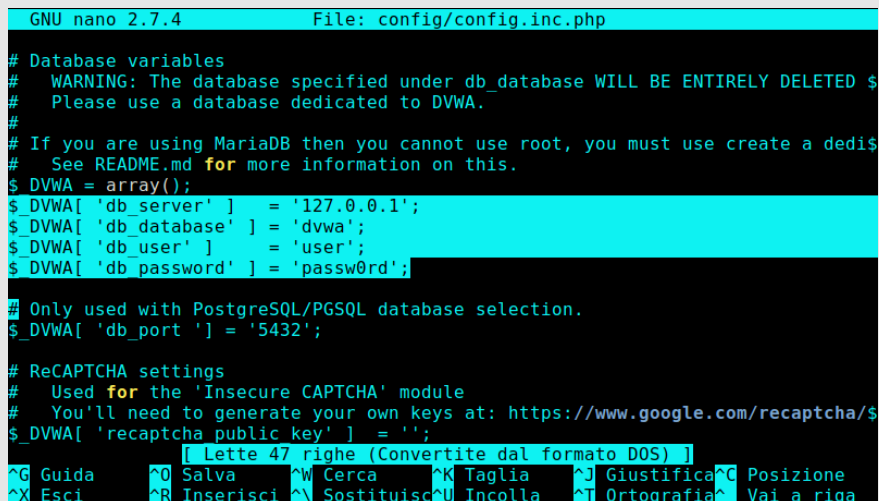



Figura X: dal sito di reCAPTCHA generiamo due chiavi (pubblica e privata) che andremo ad inserire nel file di configurazione.

A questo punto possiamo modificare il file creato precedentemente:

```
$ nano config/config.inc.php
```

Troviamo le righe di configurazione nel file e sostituiamole con le coordinate in nostro possesso:

```
$_DVWA[ 'db_server' ]   = '127.0.0.1';
$_DVWA[ 'db_database' ] = 'dvwa';
$_DVWA[ 'db_user' ]     = 'user';
```

```
$_DVWA[ 'db_password' ] = 'passw0rd';
```

Per sicurezza prendi in riferimento la screen in figura X:0

```
  GNU nano 2.7.4              File: config/config.inc.php

# Database variables
#   WARNING: The database specified under db_database WILL BE ENTIRELY DELETED $
#   Please use a database dedicated to DVWA.
#
# If you are using MariaDB then you cannot use root, you must use create a dedi$
#   See README.md for more information on this.
$_DVWA = array();
$_DVWA[ 'db_server' ]   = '127.0.0.1';
$_DVWA[ 'db_database' ] = 'dvwa';
$_DVWA[ 'db_user' ]     = 'user';
$_DVWA[ 'db_password' ] = 'passw0rd';

# Only used with PostgreSQL/PGSQL database selection.
$_DVWA[ 'db_port '] = '5432';

# ReCAPTCHA settings
#   Used for the 'Insecure CAPTCHA' module
#   You'll need to generate your own keys at: https://www.google.com/recaptcha/$
$_DVWA[ 'recaptcha_public_key' ]  = '';
                [ Lette 47 righe (Convertite dal formato DOS) ]
^G Guida      ^O Salva      ^W Cerca      ^K Taglia     ^J Giustifica^C Posizione
^X Esci       ^R Inserisci  ^\ Sostituisc^U Incolla     ^T Ortografia^_ Vai a riga
```

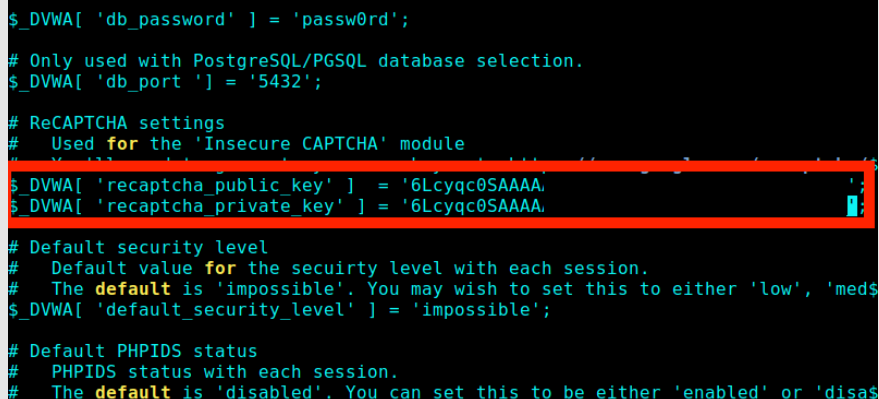



Figura X: effettua le modifiche, quindi salva con CTRL+X, S e INVIO.

Se abbiamo creato le chiavi reCAPTCHA, aggiungiamole nella seguente porzione:

```
$_DVWA[ 'recaptcha_public_key' ]  =
'6Lcyqc0SAAA********************';

$_DVWA[ 'recaptcha_private_key' ] =
'6Lcyqc0SAAAAAH****************';
```

Sempre per sicurezza, prendi in riferimento la screen in figura X:

```
$_DVWA[ 'db_password' ] = 'passw0rd';

# Only used with PostgreSQL/PGSQL database selection.
$_DVWA[ 'db_port '] = '5432';

# ReCAPTCHA settings
#   Used for the 'Insecure CAPTCHA' module
$_DVWA[ 'recaptcha_public_key' ]  = '6Lcyqc0SAAAA                    ';
$_DVWA[ 'recaptcha_private_key' ] = '6Lcyqc0SAAAA                    ';

# Default security level
#   Default value for the secuirty level with each session.
#   The default is 'impossible'. You may wish to set this to either 'low', 'med$
$_DVWA[ 'default_security_level' ] = 'impossible';

# Default PHPIDS status
#   PHPIDS status with each session.
#   The default is 'disabled'. You can set this to be either 'enabled' or 'disa$
```

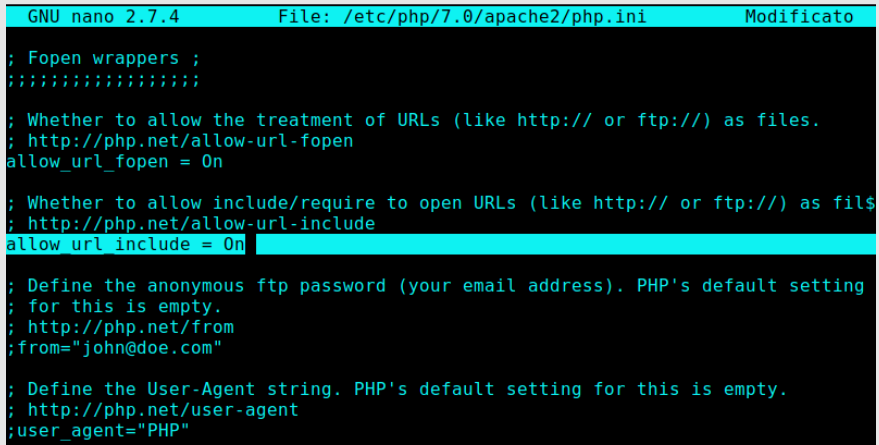
Figura X: mi raccomando, chiave pubblica e chiave privata sono due cose diverse!

Salviamo e ricolleghiamoci dalla macchina attacker all'indirizzo http://victim/vuln [figura X]:

Figura X: se tutto è stato fatto correttamente otterremo la pagina di login principale.

3.9.1.3 INSTALLARE DVWA

Qualunque dato inserito nel login non produrrà alcun risultato: il motivo è che il CMS deve essere ancora installato! Verremo catapultati in una pagina (http://victim/vuln/setup.php) in cui è presente un README e alcuni requisiti fondamentali non ottenuti, per intenderci quello che è presente in figura X:

Figura X: in questa pagina sono elencate alcune cose da fare prima di poter proseguire.

Procediamo ad abilitare alcune funzioni per rendere la situazione un po' più interessante :)

Abilitare funzioni PHP

Nel nostro caso è necessario abilitare solo "allow_url_include" ma non toglie che alcuni setup prevedano anche altri moduli disabilitati: per abilitarli è necessario modificare il file /etc/php/{versione PHP}/apache2/php.ini impostando il valore su On come in questo caso:

```
$ nano /etc/php/7.0/apache2/php.ini
```

Identifichiamo "allow_url_include" nel file (puoi usare la combinazione CTRL+W per cercare nel file) e impostiamo il valore su On come in figura X:

```
  GNU nano 2.7.4          File: /etc/php/7.0/apache2/php.ini          Modificato

; Fopen wrappers ;
;;;;;;;;;;;;;;;;;;

; Whether to allow the treatment of URLs (like http:// or ftp://) as files.
; http://php.net/allow-url-fopen
allow_url_fopen = On

; Whether to allow include/require to open URLs (like http:// or ftp://) as fil$
; http://php.net/allow-url-include
allow_url_include = On

; Define the anonymous ftp password (your email address). PHP's default setting
; for this is empty.
; http://php.net/from
;from="john@doe.com"

; Define the User-Agent string. PHP's default setting for this is empty.
; http://php.net/user-agent
;user_agent="PHP"
```

Figura X: cerca il valore con CTRL+W, quindi salva il file con CTRL+X, S e INVIO.

Riavviamo il web server Apache2:

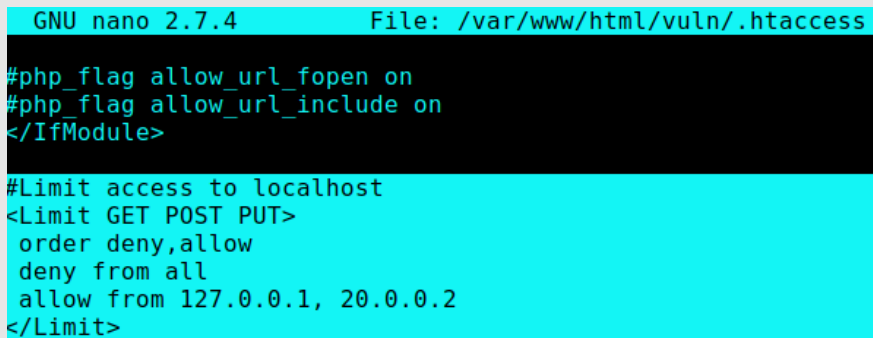
```
$ service apache2 restart
```

Imposta i permessi alle cartelle e file

Per sfruttare tutte le caratteristiche impostiamo i permessi per consentire la scrittura al programma:

```
$ chmod -R -f 777 /var/www/html/vuln/hackable/uploads/
$ chmod 777 /var/www/html/vuln/external/phpids/0.6/lib/IDS/tmp/phpids_log.txt
$ chmod -R -f 777 /var/www/html/vuln/config
```

Abilitare la Whitelist

Non è proprio saggio mettere in rete una web app vulnerabile, pronta per essere bucata da chiunque, non trovi? Possiamo allora modificare il file .htaccess[1] e indicare solo quali IP possono collegarsi al portale:

```
$ nano .htaccess
```

[1] Maggiori informazioni al capitolo X

quindi identifichiamo "Limit access to localhost", decommentiamo tutte le righe eliminando gli asterischi e aggiungendo alla voce "allow from" l'indirizzo IP della macchina attacker:

```
#Limit access to localhost
<Limit GET POST PUT>
 order deny,allow
 deny from all
 allow from 127.0.0.1, 20.0.0.2
</Limit>
```

Il risultato sarà identico a quanto mostrato in figura X.

```
  GNU nano 2.7.4          File: /var/www/html/vuln/.htaccess

#php_flag allow_url_fopen on
#php_flag allow_url_include on
</IfModule>

#Limit access to localhost
<Limit GET POST PUT>
 order deny,allow
 deny from all
 allow from 127.0.0.1, 20.0.0.2
</Limit>
```

Figura X: rimuoviamo gli asterischi, aggiungiamo l'ip 20.0.0.2 quindi salviamo con CTRL+X, S e INVIO.

Avviare il setup di DVWA

A configurazione finita possiamo installare il CMS tramite il pulsante "Create / Reset Database" in fondo. Verremo rendirizzati alla pagina da login, dove tutto ha inizio.

Rieffettuiamo il login (username: "admin", password: "password) e impostiamo il Security Level su Low [figura X]. In questo modo potremo effettuare tutti i test che vorremo ai danni della macchina.

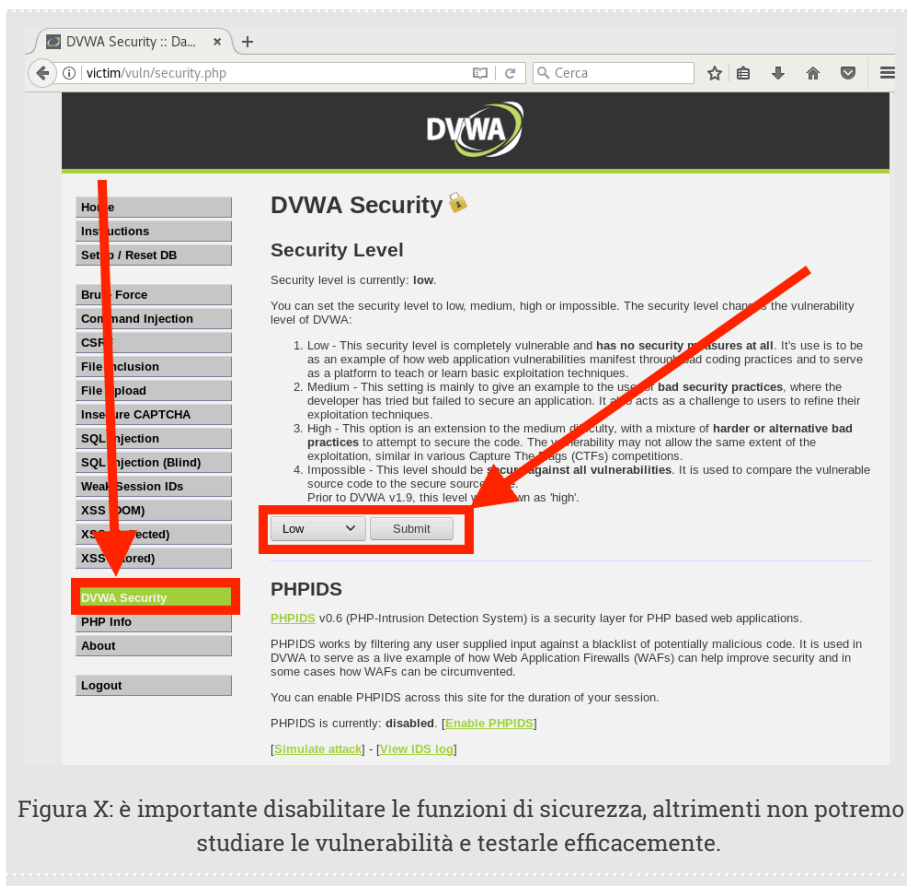



Figura X: è importante disabilitare le funzioni di sicurezza, altrimenti non potremo studiare le vulnerabilità e testarle efficacemente.

3.10 Oltre i fondamentali

Tutto quello che abbiamo visto è in realtà una piccolissima parte di quanto è presente nel World Wide Web. Avremmo potuto parlarne all'infinito (la tecnologia non si ferma mai!) ma ci siamo limitati a vedere solo quello che ti permette di metter mani al 90% di quanto ti "passerà tra le mani" e di effettuare i tuoi attacchi senza andare alla cieca.

In programmazione web, come nella vita reale, però ognuno è libero di progettare il software come meglio crede. È doveroso quindi ricordarsi che:

- Non tutti usano un server con Linux

- Non tutti progettano i loro back-end in PHP
- Non tutti usano assiduamente SQL per salvare i loro dati

Già questi tre punti, presi singolarmente, farebbero testo ad altri altrettanti volumi; inoltre i fondamentali spiegati non basterebbero a ricoprire gli attacchi "marginali" che è possibile effettuare. Il Web è tutto un mondo da scoprire e tu ci sei dentro!